Алан Купер
Психбольница в руках пациентов

Алан Купер
Психбольница в руках пациентов
Почему высокие технологии сводят нас с ума и как восстановить душевное равновесие

Посвящается Сью, Скотт и Мери с любовью

Отзывы на книгу Алана Купера

От этого становится не по себе, но это правда. Персональные компьютеры породили новую взаимозависимость Новой Эры. Они позорят нас, раздражают нас, но мы продолжаем тратить на них деньги. Книга Алана Купера объясняет, почему все должно быть не так и что мы можем сделать для исправления ситуации. Чтение доставляет удовольствие и помогает спуститься с небес на землю.

– Жан-Луи Гассе, президент Be, Inc.

И впрямь пациенты. Людям пора проснуться и сказать «Все, с нас хватит!» Снова Алан Купер освещает нам путь. Чтение его книг следует сделать обязательным во всех компаниях, имеющих отношение к высоким технологиям и считающих, что они работают на благо клиентов. Нам нужно больше книг, подобных этой, больше людей, похожих на Алана Купера.

– ДонНорман, Nielsen Norman Group; автор книги «The Invisible Computer»

Если вы когда-нибудь задумывались, почему программы, написанные для компьютеров, и многочисленные электронные штучки ведут себя так, словно были созданы болваном или пытаются сделать болвана из вас, прочтите эту книгу. Книга Купера – это манифест о том, как повысить качество жизни в век, когда мы проводим много времени во взаимодействии с технологией.

– Питер Хиршберг, президент Elemental Software

Подход к проектированию, предлагаемый Купером, быстро завоевывает последователей, среди которых и такие клиенты, как Sun Microsystems, Coca-Cola, Compaq и Dow Jones.

– Fast Company Magazine

Целеориентированная методология проектирования Алана Купера, сосредотачивающая внимание на конкретных конечных пользователях (персонажах), привела команду проектировщиков к элегантному решению невероятно сложной проблемы проектирования пользовательского интерфейса.

– Карен Ивенсон, руководитель инженерных разработок Sony Trans Com

Браво! Живой и проникновенный трактат о проектировании взаимодействия от того, кто стоит у истоков. Раздавайте экземпляры друзьям, коллегам, своим клиентам. Об этой книге обязательно будут говорить, а возможно, и спорить.

– Клемент Мок, основатель Archetype Studio, креативный директор Sapient

Эта книга изменит ваши отношения с технологией, будь вы ее создатель или потребитель. Технология призвана помогать человеку, а не приводить его в замешательство. Это лучшая из прочитанных мною книг о проектировании взаимодействий: в ней Алан рассказывает, почему компьютеры неверно взаимодействуют с людьми и как излечить эту хроническую болезнь. Обязательная книга для всех, кто имеет дело с продуктами высоких технологий.

– Джефф Хадфилд, главный редактор Visual Basic Programmer's Journal

Мы находимся под сильным впечатлением от энтузиазма, профессионализма и методов работы нескольких команд Cooper Interaction Design, сотрудничающих с нами над перепроектированием центральных модулей R/3.

– Маттиас Беринг, руководитель программы, SAP

Если бы Алан Купер присутствовал на сказочном шествии короля через город, то первым бы заявил о том, что король голый, после чего показал бы всем, как создавать наряды привлекательные, экономичные и приятные в ношении. В этой книге Купер бросает вызов индустрии программного обеспечения на всех уровнях – от тех, кто пишет код, до высшего руководства, демонстрируя изъяны современных программ и указывая на приемы проектирования, позволяющие исправить положение. Основываясь на своем богатом опыте в проектировании приложений, он излагает материал настолько ясно и практично, что книга может стать бесценным откровением для любого читателя – даже того, чей контакт с индустрией компьютерных приложений ограничен использованием продуктов, получивших название «танцующих медведей».

– Терри Виноград, профессор информатики Стэнфордского университета

Отличные программы создаются для пользователей. Работать с такими программами легко и приятно. Такая работа эффективна. Алан Купер четко показывает, что в массе своей программы создаются для программистов: на основе их догадок о потребностях пользователя, из любимых функций, индивидуальных особенностей стиля. Эта умная книга учит руководителей тому, что они должны знать, чтобы создавать системы, покоряющие рынок. Любой участник рынка – и самый удачливый и всего добившийся, и только еще полный надежд стать следующей сверхновой, поймет, что эта книга – одна из наиболее глубоких, практичных и полезных.

– Ларри Кили, президент Doblin Group

Сочетание острого ума автора и его высочайшего профессионализма делают эту книгу не только очень познавательной, но еще и весьма увлекательной.

– Хайди Ройзен, Be, Inc.

Алан Купер понимает разницу между интерфейсом и взаимодействием лучше, чем кто-либо из знакомых мне людей. Его идеи опираются на многолетний опыт в содействии разработке продуктов, элегантно и ненавязчиво проникающих в нашу жизнь. Он претворял свои идеи в жизнь много лет и теперь, наконец, нашел время превратить свой практический опыт в четкое описание задачи, с которой мы столкнулись, в методологию побега из психбольницы, которую мы с такой любовью выстроили. Читайте дальше – и обретете свободу.

– Пол Саффо, директор Института Будущего

Об авторе

По мнению Митча Уэйта, основателя Waite Group Press, Алан Купер – «Отец Visual Basic». Перу Алана Купера принадлежит книга «About Face: The Essentials of User Interface Design» (IDG Books, 1995). В 1994 году Билл Гейтс представил Алана к редкой и желанной награде Windows Pioneer Award, признав, что его вклад в изобретение Visual Basic способствовал успеху системы Microsoft Windows. В 1998 году Алан был представлен к награде Software Visionary Award. Сегодня он возглавляет Cooper Interaction Design, консультирующую фирму, которая выполняла работы по проектированию продуктов для таких компаний, как 3M, Elemental, Ericsson, Fujitsu, IBM, Logitech, McGraw-Hill, Sagent, SAP, Sony, Varian, VISA и Sun Microsystems.

Алан открыто выступает в защиту тех, о ком забывают в процессе разработки электронных продуктов, – покупателей.

В течение двадцати лет Алан Купер проектировал и создавал потребительские программные продукты, среди которых SuperProject, MicroPhone II для Windows, а также интерфейс визуального программирования для Visual Basic. В 1976 году Купер основал компанию Structured Systems Group Inc., которая, как сказано в книге «Fire In the Valley»

(Пожар в Кремниевой Долине), выпустила «возможно, первое серьезное бизнес-приложение для микрокомпьютеров».

Купер состоит в организациях Corporate Design Foundation и American Center for Design. Он бывший директор калифорнийского отделения ассоциации проектирования программного обеспечения (Association for Software Design), а также член совета директоров этой организации. Купер – директор Software Design и Software Forum, а также основатель SEF Windows SIG, крупнейшей в мире группы разработчиков для Windows. Он часто и уверенно выступает как делегат индустрии и пишет о пользовательском интерфейсе и концептуальном проектировании программных продуктов.

Благодарности

Я не смог бы написать эту книгу без помощи и участия многих замечательных друзей и коллег. В частности, несколько человек выполнили трудную и кропотливую работу – прочитали и прокомментировали рукопись, иногда по несколько раз. Их комментарии вынуждали меня отвечать на сложные вопросы, предлагать читателю новые темы, резюмировать свои соображения, умиротворяли мой пыл, усмиряли мое дикое негодование. Эта книга стала гораздо лучше благодаря вкладу Ким Гудвин (Kim Goodwin), Лейн Хэлли (Lane Halley), Келли Боумен (Kelly Bowman), Скотта Мак-Грегора (Scott McGregor), Дэвида Уеста (David West), Майка Нельсона (Mike Nelson), Марка Джирска (Mark Dziersk), Алана Карпа (Alan Karp), Терри Суок (Terry Swack), Луи Вейцмана (Louie Weitzman), Уэйна Гринвуда (Wayne Greenwood), Райана Ольшавски (Ryan Olshavsky), Джона Майера (John Meyer), Лайзы Сондерс (Lisa Saunders), Винни Шоуз (Winnie Shows), Кевина Вандрайка (Kevin Wandryk), Глена Халстеда (Glenn Halstead), Брайана О'Салливана (Bryan O'Sullivan), Чака Оуэна (Chuck Owen), Майка Суэйни (Mike Swaine), а также Скипа Уолтера (Skip Walter). Спасибо вам за время, участие и мудрость. Особенно ценными для меня в плане кристаллизации тем книги стали комментарии и советы Джонатана Кормана (Jonathan Korman). Я должен также поблагодарить всех талантливых и усердных сотрудников CID[1], которые делали за меня мою работу, пока я был занят книгой. Особой благодарности заслуживает Уэйн Гринвуд (Wayne Greenwood), который замечательно работал под жестким прессингом, сохраняя наш боевой дух и высокое качество наших проектов.

Создание иллюстраций стало одной из наиболее интересных задач. Чед Кубо (Chad Kubo), мастер изобразительного искусства, проделал замечательную работу – превратил мои расплывчатые идеи в четкие, запоминающиеся образы. Они многое привнесли в книгу. И эти иллюстрации никогда не появились бы без неутомимого художественного руководства со стороны Пенни Бейлес (Penny Bayless) и Дэвида Хейла (David Hale). Были и другие люди, решавшие производственные задачи. Спасибо Брит Кацен (Brit Katzen) за изучение и перепроверку фактов, спасибо Майку Генри (Mike Henry) за литературное редактирование.

Создание книги – это деловое предприятие, и за то, что оно стало успешным, я должен поблагодарить команду сведущих в технологии предпринимателей, возглавляемую моим агентом, Джимом Левайном (Jim Levine), и включающую Глена Халстеда (Glenn Halstead), Линн Боумен (Lynne Bowman), Келли Боумен (Kelly Bowman), а также Сью Купер (Sue

[1] Компания Cooper Interaction Design.

Cooper). Со стороны издательства Macmillan поддержку на протяжении всего проекта осуществлял Брэд Джоунс (Brad Jones), однако наибольшего внимания заслуживает Крис Вебб (Chris Webb), чья целеустремленность, сосредоточенность и тяжелый труд сделали возможным создание этой книги.

Я очень ценю людей, оказавших мне моральную поддержку, предоставивших реальные истории, советы и потративших свое время. Большое спасибо Дэниелу Эпллману (Daniel Appleman), Тодду Баше (Todd Basche), Крису Байеру (Chris Bauer), Джеффу Безо (Jeff Bezos), Элис Блэр (Alice Blair), Мишель Борк (Michel Bourque), По Бронсону (Po Bronson), Стиву Калде (Steve Calde), Дэвиду Карлику (David Carlick), Джеффу Карлику (Jeff Carlick), Кэрол Кристи (Carol Christie), Клэй Коллье (Clay Collier), Кендаллу Косби (Kendall Cosby), Дэну Крэйну (Dan Crane), Роберту Крингели (Robert X. Cringely), Трою Дэниелсу (Troy Daniels), Лайзе Пауэрс (Lisa Powers), Филипу Энглхарту (Philip Englehardt), Карен Ивенсен (Karen Evensen), Риджели Эверс (Ridgely Evers), Ройял Фаррос (Royal Farros), Пэт Флек (Pat Fleck), Дэвиду Фору (David Fore), Эду Форману (Ed Forman), Эду Фридкину (Ed Fredkin), Жану-Луи Гассе (Jean-Louis Gassee), Джиму Гею (Jim Gay), Рассу Голдину (Russ Goldin), Владу Горелику (Vlad Gorelik), Марсии Грегори (Marcia Gregory), Гарренту Грюнеру (Garrett Gruener), Чаку Хартледжу (Chuck Hartledge), Теду Харвуду (Ted Harwood), Уиллу Хирсту (Will Hearst), Тамре Хитершоу-Харт (Tamra Heathershaw-Hart), ДжейДи Хильдебранду (JD Hildebrand), Лори Хиллз (Laurie Hills), Питеру Хиршбергу (Peter Hirshberg), Ларри Кили (Larry Keeley), Гэри Краткину (Gary Kratkin), Деборе Курата (Deborah Kurata), Тому Лефлеру (Tom Lafleur), Полу Лотону (Paul Laughton), Элен Леви (Ellen Levy), Стивену Листу (Steven List), ТиСи Мангану (TC Mangan), Дэвиду Майстеру (David Maister), Роберту Мэю (Robert May), Дону Мак-Кини (Don McKinney), Кэтрин Медоуз (Kathryn Meadows), Лайзе Митчелл (Lisa Mitchell), Джеффри Муру (Geoffrey Moore), Брюсу Мовери (Bruce Mowery), Нату Майерсу (Nate Myers), Эду Нихаусу (Ed Niehaus), Констанс Петерсен (Constance Petersen), Кейту Плису (Keith Pleas), Роберту Райнману (Robert Reimann), Джону Ривлину (John Rivlin), Говарду Рейнгольду (Howard Rheingold), Гейди Ройцену (Heidi Roizen), Нилу Рубенкингу (Neil Rubenking), Полу Сафо (Paul Saffo), Джошу Зайдену (Josh Seiden), Рассу Зигельману (Russ Siegelman), Донне Слоут (Donna Slote), Линде Стоун (Linda Stone), Тони Уокеру (Toni Walker), Кевину Уиксу (Kevin Weeks), Кевину Уэлшу (Kevin Welch), Дэну Уиллису (Dan Willis), Хэзер Уинкль (Heather Winkle), Стефану Уайлдстрому (Stephen Wildstrom), Терри Винограду (Terry Winograd), Джону Цикеру (John Zicker) и Пьерлуиджи Заппакосте (Pierluigi Zappacosta).

Этот «проект на год» продлился двадцать месяцев, и моя семья проявила большое терпение. На мне величайший долг любви и благодарности перед моей супругой Сью Купер и моими красивыми юными сыновьями Скоттом и Марти. Люблю вас всем сердцем.

Предисловие научного редактора

От своего лица благодарю издательство «Символ-Плюс» за выпуск этой в высшей степени актуальной книги.

Молодая, бурно развивающаяся отрасль информационных технологий (ИТ) все глубже проникает в нашу жизнь. Стремительно расширяется аудитория потребителей электронных продуктов: например, мобильными телефонами сейчас пользуются дети и пожилые люди, хотя совсем недавно этот продукт относился к разряду элитных. Но, к сожалению, большинство производителей до сих пор делают свои товары по принципу «главное – быстрее продать». При этом основными конкурентными преимуществами продукта и продавец и покупатель, как правило, считают его функциональные возможности, иными словами – «навороченность». На рынке появляются сотни моделей сотовых телефонов со встроенными камерами и цифровыми проигрывателями. В то же время отсутствуют телефоны с крупными кнопками, большим экраном, увеличенным размером шрифта – именно такой телефон многие хотели бы купить для своих немолодых родителей, а может

быть, и для себя. Неудобные, сложные продукты окружают нас, и этот круг становится все более тесным.

Подобная ситуация сложилась не только на западном рынке, о котором пишет Алан Купер, но в еще большей степени и на российском, где действуют дополнительные факторы, усугубляющие проблему.

Во-первых, дешевизна многих рабочих мест, например кассиров супермаркета или работников операторских центров (call-centres), снижает мотивацию работодателя, перестающего требовать от программных продуктов высоких эргономических характеристик. Часто работодатель нанимает несколько дополнительных сотрудников вместо того, чтобы вложить средства в повышение эффективности труда уже работающих.

Во-вторых, менталитет российского человека, с раннего возраста привыкшего бороться с неудобными в быту вещами, запрещает ему жаловаться на такие «мелочи», как нелогичность работы устройства или бессмысленность сообщений об ошибках, выдаваемых программой. Жалобы на что-то для наших соотечественников равносильны признанию в собственной слабости и глупости. Они не пеняют на сложные бытовые условия и на неудобство тех или иных продуктов; в крайнем случае, они пытаются обойти проблему или просто игнорируют ее. Это порождает скрытый спрос на приятные в использовании продукты, о наличии которого большинство разработчиков информационных систем даже не догадывается.

В-третьих, многие заказчики до сих пор живут сегодняшним днем, не задумываясь над тем, что вложения в эргономику – как в виде дополнительных требований к поставщику программных продуктов, так и в виде средств, потраченных на грамотное проектирование пользовательское интерфейса, – вернутся ощутимой прибылью. Производители, в свои очередь, предпочитают работать «по старинке» и не предлагать заказчику полноценное проектирование пользовательского интерфейса на начальном этапе работы, еще до набора команды программистов или выбора программной платформы. Нередко главная цель состоит в том, чтоб «отхватить» проект и поскорее начать его делать, надеясь, что в процессе все проблемные места уж как-нибудь сами «утрясутся». Обучать, воспитывать заказчика, помогать ему и его бизнесу в долгосрочной перспективе – непопулярное дело в среде разработчиков. И, честно говоря, не всегда благодарное. По собственному опыту знаю, что при разработке пользовательских интерфейсов значительные ресурсы расходуются именно на воспитание разработчиков и заказчиков продуктов.

Несмотря на перечисленные выше факторы, в последние год-два ситуация быстро меняется: благодаря росту экономики работодателю теперь выгоднее вкладывать средства в уже работающих сотрудников, чем нанимать новых; растет искушенность российских потребителей, а, следовательно, и их требования к покупаемым товарам; многие сегменты рынка ИТ развиваются, и эргономическая составляющая проектов становится конкурентным преимуществом для компаний-разработчиков.

Алан в своей неподражаемой, провокационной манере критикует устаревшие методы создания программных продуктов. Он предлагает новый революционный способ разработки, результатом которого станут востребованные пользователями продукты. Можно без преувеличения сказать: подход Купера задает новый вектор развития индустрии ИТ – индустрии облегчающей, а не усложняющей нам жизнь.

Первое издание книги вышло пять лет назад (1999). С этого момента многие ведущие компании, например Microsoft, SAP, Oracle, взяли предлагаемые Купером методы на вооружение. Так, Microsoft использует персонажей при разработке Longhorn.

Я надеюсь, что российский рынок ИТ также услышит голос Алана Купера

Алексей Копылов,
ведущий специалист компании UIDesign Group,
разрабатывающей пользовательские интерфейсы

Предисловие

Спасайся, кто может – началось вторжение компьютеров. Компьютеров, мощность которых приводит в трепет и которые решают все более важные задачи посредством неуклюжих, старомодных интерфейсов. Проникая во всевозможные сферы нашей жизни, компьютеры будут раздражать нас, приводить в ярость, а кое-кого даже убьют. В свою очередь, мы испытаем соблазн уничтожить свои компьютеры, но не посмеем, потому что и сегодня уже полностью, необратимо зависим от этих многообещающих монстров, делающих современную жизнь возможной.

К счастью, у нас есть другой выход. Необходимо коренным образом изменить представления о взаимодействии людей и машин. Изменить эти отношения на глубинном уровне, создать не существовавшие ранее подходы, ибо причина наших растущих проблем не в машинах, а в нас самих. Люди создали ненавистные интерфейсы; люди продолжают использовать некорректно функционирующие машины, мирясь с нагрузкой на глаза, создаваемой неуклюжими интерфейсами, мирясь с болью в спине и разрушением сухожилий в запястьях. Как подсказывает название книги, все мы пациенты технопсихушки, которую сами же создали.

Эта книга открывает путь к спасению. Вернее, Алан Купер показывает, что дверь на свободу широко распахнута. Мы можем уйти в любой момент, но в своем безумии мы до сих пор этого не замечали. Секрет в том, чтобы переопределить способ взаимодействия с компьютерами в более широком контексте.

Алан Купер – не просто один из пациентов, он еще и отступник, чьи идеи, вероятно, приведут в ярость тех, кто хочет держать нас под замком – инженеров, создавших ненавистные нам системы и по-прежнему считающих совершенствование интерфейсов выходом из этого кошмара. Но само понятие интерфейса – пережиток эры, когда компьютеры встречались редко и были слабыми, когда они почти не были способны взаимодействовать со своими хозяевами, людьми. Интерфейсы имели смысл, когда все взаимодействие происходило в неизведанном мире за стеклянным экраном компьютера. Сегодня же они стали просто опасными, потому что мы живем в мире, где компьютеры проникают во все уголки жизни. Компьютеры более не соединяются с людьми посредством интерфейсов, но взаимодействуют с нами, и это взаимодействие будет постепенно углубляться, становиться более тонким и более критичным для нашего душевного равновесия и, в конечном итоге, выживания.

Алан Купер понимает разницу между интерфейсом и взаимодействие лучше, чем кто-либо из знакомых мне людей. Его идеи опираются в многолетний опыт в содействии разработке продуктов, элегантно и ненавязчиво проникающих в нашу жизнь. Он претворял свои идеи в жизнь много лет и теперь, наконец, нашел время превратить свой практически опыт в четкое описание задачи, с которой мы столкнулись, в методологию побега из психбольницы, которую мы с такой любовью выстроил! Читайте дальше – и обретете свободу.

Пол Саффо (Paul Saffo),
директор Института Будущего

Введение

Книга-обоснование

Я собирался написать совсем другую книгу – книгу-руководство о процессе проектирования взаимодействия. В мае 1997 года во время поездки в Тоскану двое моих друзей, Дон Мак-Кини (Don McKinney) и Дэйв Карлик (Dave Carlick), уговорили меня написать книгу, которую вы держите в руках. Они убедили меня, что следует, прежде всего, обращаться к деловой аудитории.

Они знали, что я хотел написать руководство, и, поощряя эту идею, все-таки выражали

сомнение в необходимости книги о проектировании взаимодействия (interaction design). Они хотели, чтобы я написал книгу, убеждающую в ценности этого процесса. Аргумент, конечно, занимательный, но я не был уверен, что смогу.

Однажды поздней ночью на веранде нашей общей желтой виллы над Фиренце у меня состоялся серьезный разговор с Дэйвом и Доном. На столе несколько пустых бутылок из-под кьянти и остатки хлеба, сыра и оливок. Сияют звезды, светлячки порхают над лужайкой, а вдалеке подмигивают огни древних куполов столицы Тосканы. Дэйв снова предлагает мне отложить идею книги-руководства и вместо этого «дать деловое обоснование проектированию взаимодействия».

Я энергично протестую: «Дэйв, я же не знаю, как писать такую книгу». И начинаю загибать пальцы: «Мне придется объяснять, насколько отвратителен существующий процесс разработки, как компании теряют деньги на неэффективном создании программного обеспечения, как ненадежны неудовлетворенные клиенты и как все эти проблемы может разрешить более совершенный процесс проектирования».

Дэйв перебивает меня: «Алан, это называется главами».

Эта реплика лишает меня воли. Я вдруг осознаю, что повторяю избитый сценарий и что Дэйв прав. Книга, содержащая «бизнес-обоснование», необходима и будет более своевременна, чем книга-руководство. Дэйв и Дон убедили меня, что я действительно способен написать такую книгу.

Инженер, сведущий в бизнесе, либо бизнесмен, сведущий в технологии

Успешный профессионал XXI века – это либо инженер, сведущий в бизнесе, либо бизнесмен, сведущий в технологии, и именно такому человеку я адресую свою книгу.

Бизнесмен, сведущий в технологии, осознает, что его успех зависит от качества доступной ему информации и изощренности, с которой он сможет эту информацию использовать. С другой стороны, инженер, сведущий в бизнесе, – это предприимчивый конструктор или ученый, специализирующийся на технологии, но при этом обладающий острой деловой хваткой и осознанием силы информации. Оба новых архетипа будут доминировать в современном бизнесе.

Всех деловых людей можно разделить на две категории: на тех, кто овладеет высокими технологиями, и тех, кто скоро покинет деловую арену. Руководящий работник уже не может делегировать обработку информации специалистам, ведь бизнес и есть обработка информации. Сегодня вы можете выделиться качеством своих систем обработки информации, но не качеством систем производства. Если вы производите что-либо, то весьма вероятно, что продукт содержит микросхему. Если вы предлагаете услугу, то, скорее всего, вы используете компьютеризованные инструменты. Попытка выявить предприятия, зависящие от высоких технологий, столь же глупая затея, как попытка выявить предприятия, зависящие от телефона. Революция высоких технологий затронула все сферы деловой деятельности, а информация в цифровом формате стала основой вашего рабочего дня.

Кто-то сказал: «Человеку свойственно ошибаться, но чтобы провалить дело капитально, необходим компьютер». Неэффективные механические системы способны приносить убытки в пару центов на каждой производимой детали, однако из-за некачественных информационных процессов можно потерять целую компанию. Влияние на вашу компанию продуктов, основанных на программном обеспечении, как и влияние инженеров, создающих эти продукты, огромно.

Жаль, что наши цифровые инструменты сложны для изучения и понимания, сложны в применении; они часто препятствуют достижению наших целей. Мы теряем деньги, время, возможности. Будучи инженером, сведущим в бизнесе, или же бизнесменом, сведущем в технологии, вы создаете или потребляете продукты, основанные на программном обеспечении, а скорее, потребляете и создаете одновременно. Обладание более совершенными, более простыми в освоении и применении высокотехнологичными

продуктами – в ваших личных и профессиональных интересах. Более качественные продукты не требуют больших затрат времени на создание, и их создание не обходится дороже. Ирония в том, что они не должны быть сложными и таковыми являются лишь потому, что устарели процессы конструирования, и эти процессы требуют ремонта. Лишь давние традиции, основанные на заблуждениях, препятствуют сегодня получению более качественных продуктов. В книге показано, как можно требовать и получать лучшие продукты, которых вы заслуживаете.

Идея этой книги не сложна: мы можем создавать мощные и приятные продукты, основанные на программном обеспечении, используя простой прием: *проектируя* взаимодействие продукта с пользователем *до* этапа программирования. Сегодня мы этого не делаем, несмотря на распространенное мнение. Проектирование интерактивных продуктов, использующих программное обеспечение, – специализация столь же сложная, как собственно разработка.

* * *

Сделав выбор в пользу книги-обоснования и отказавшись от книги-руководства, я молю о прощении каждого проектировщика взаимодействия, которому придется читать эту книгу. Из почтения к деловой части аудитории она содержит лишь краткие отступления в область практической методологии проектирования взаимодействий (в основном в главах части IV «Проектирование взаимодействий – выгодный бизнес»). Информации достаточно, чтобы показать, что такая методология существует, что она применима в любой предметной области и что ее преимущества очевидны всем, независимо от компетенции в технической области.

Недавно я познакомился с руководителем высокого ранга одной из крупнейших технологических компаний мира. Официальное название его должности – вице-президент по вопросам удобства использования продуктов (юзабилити), и он несет ответственность за очень многие программные продукты, как небольшие, так и крупные. Это выдающийся и состоявшийся человек, выходец из сообщества HCI (Human-Computer Interaction, взаимодействие человека и компьютера), где принят формализованный подход. Он, как и его компания в целом, искушен в «эргономике» – в тестировании и наблюдении из-за односторонних зеркал. Однако он пришел говорить со мной не о тестировании, а о проектировании, и не о пользователях, а о персонажах. Он рассказал, что в его компании полностью прекращено эргономическое тестирование (называемое в этой книге юзабилити – тестированием) на завершающей стадии разработки; вместо этого усилия прикладываются до начала разработки – при проектировании. Более того, он упомянул, что все его люди, умеющие и привыкшие наблюдать за пользователями в искусственных условиях, проходят переподготовку и будут заниматься этнографическими исследованиями в «полевых» условиях.

Этот руководитель и его компания – символ тех огромных изменений, которые произошли в отрасли за пять коротких лет с момента выхода в 1999 году первого издания книги «The Inmates Are Running the Asylum» («Психбольница в руках пациентов»). Книга послужила одновременно манифестом для революции и учебником для науки. Не сосчитать, сколько писем я получил от менеджеров среднего звена, в которых рассказывалось, как после прочтения книги они покупали по экземпляру каждому из вышестоящих руководителей. Между тем разработчики программного обеспечения и университеты восприняли три главы части IV «Проектирование взаимодействий – выгодный бизнес» как исходный материал для руководства «сделай сам» по претворению в жизнь целеориентированного (Goal-Directed®) проектирования при помощи персонажей.

Я весьма признателен всем менеджерам, программистам, руководителям и практикам эргономики за то, что они воспользовались идеями этой книги и вывели удобные программы из лабораторий в жизнь, сместили фокус с тестирования на проектирование. Благодаря их

усилиям полностью изменился ландшафт профессии эргономиста. Сегодня в большинстве организаций, с которыми мне приходится иметь дело, на полной ставке работает один или несколько профессиональных проектировщиков взаимодействия, и влияние этих людей на качество и поведение программных продуктов и услуг постоянно растет. Мне приятно осознавать, что эта книга способствовала их успеху.

Мне вспоминается, как я читал программную речь на конференции программистов в 1999 году, вскоре после опубликования книги. Название речи совпадало с названием книги, и речь я начал с утверждения, что «психбольница находится в руках пациентов, и эти пациенты – вы». В этот момент, когда более 2500 инженеров пытались воспринять мое обвинение, можно было услышать, как летит муха. В молчании, охватившем зал, я продолжил представление основных посылок книги, и час спустя толпа хомо логикус настолько прониклась моими аргументами, что аплодировала стоя. Удивительно, но большинство программистов с энтузиазмом восприняли идеи проектирования и проектировщиков взаимодействия. Они понимают, что нуждаются в содействии там, где речь идет о человеческой стороне конструирования программ, и счастливы получить, наконец, некоторые полезные указания. Программисты признают, что любая практика, повышающая качество и успех программ, не является для них угрозой. В прошлом руководители считали проектирование взаимодействия задачей программирования и делегировали ее решение программистам, которые прилежно трудились, хотя их опыт, подготовка, настрой и рабочее расписание не позволяли добиться успеха. С целью диагностирования проблемы эта книга подробно описывает такой провал, который всегда оказывается провалом программиста. Некоторые из программистов обиделись на мои описания, вообразив, что я злословлю или пытаюсь возложить на программистов вину за некачественные программы. Определенно, они являются агентами в создании некачественных программ, но никоим образом не заслуживают осуждения. Я не виню программистов за сложные в использовании программы, и мне очень жаль, что у некоторых из них сложилось обратное впечатление. За некоторыми исключениями, знакомые мне программисты прилежны и добросовестны в своем желании угодить конечным пользователям, равно как и неустанны в стремлении повышать качество своих программ. Подобно пользователям, программисты – лишь жертвы неверного процесса, оставляющего слишком мало времени, дающего слишком много противоречивых приказов и слишком мало действительно ценных указаний. Мне очень жаль, если у кого-то из программистов сложилось впечатление, что я их обвиняю.

Неэластичность процесса создания программ, а в особенности высокая стоимость программирования и низкое качество взаимодействия – проблема, попросту говоря, не технического плана. Это результат применения бизнес-практик в том направлении, в котором они устарели – в разработке программ. С чистым сердцем, наилучшими намерениями и с благословения высшего руководства программисты пытаются решить эту проблему инженерным путем. Но более интенсивная или качественная работа в этом направлении не позволяет добиться успеха. Программисты ощущают все большую тщетность своих усилий, и их отчаяние нарастает.

В своих недавних путешествиях я заметил нарастающую тревогу в сообществе программистов. С прискорбием могу сообщить, что хуже всего чувствуют себя лучшие и самые опытные программисты. Они прикладывают титанические усилия, но впадают в цинизм и испытывают тоску, понимая, что их умения растрачиваются впустую. Они могут и не знать точно, как именно получается, что их квалификация не находит правильного применения, но они не могут не видеть очевидных фактов. Многие из лучших программистов вообще перестали программировать, поскольку работа раздражает их. Они ушли в преподавание, стали проповедниками, писателями, консультантами, потому что эти занятия не оставляют ощущения пустой траты времени и сил. Но этих неприятнейших потерь вполне можно избежать. (В некотором смысле движение свободных программ с открытым кодом можно назвать раем для этих отчаявшихся программистов – тут они могут

писать код по своим стандартам и быть судимыми только равными, не выслушивая советы и не имея необходимости терпеть вмешательство маркетологов или руководителей.)

Программистам не хватает времени, четких указаний или адекватных планов, позволяющих добиться успеха. Эти три вещи – на совести руководящих работников, именно они не дают всего этого программистам, причем вовсе не по недомыслию или злому умыслу, а по причинам, которые можно было бы обойти. Они просто не вооружены адекватными инструментами, позволяющими решать сложные и уникальные проблемы информационной эры. Ну вот, звучит так, будто я снова на кого-то нападаю, но на этот раз в мой прицел попали деловые люди, а не программисты. Повторюсь, чтобы решить проблему, ее следует сначала разобрать на составляющие. Я ищу решения, а не козлов отпущения.

Мудрый руководитель Питер Дрюкер (Peter Drucker) в свои девяносто два года, большую часть которых он провел, наблюдая и направляя руководителей, смотрит на эту проблему со своей, уникальной точки зрения. В недавнем интервью журналу «CIO» он упомянул о наивном оптимизме руководителей в пятидесятые и шестидесятые годы, когда компьютеры только пробились в деловой мир. Эти руководители думали, что компьютеры «окажут огромное воздействие на способы ведения бизнеса», но Дрюкер говорит, что «произошло совсем не это. Очень немногие руководители задавали вопрос: „Какая информация мне требуется, чтобы выполнять эту работу?“» Цифровые машины дали руководителям небывалые объемы данных, но лишь немногие поинтересовались, подходят ли эти данные для управления корпорацией. Образ существования бизнеса менялся очень быстро, однако менеджмент при этом не менялся. Дрюкер атакует наши устаревшие бухгалтерские системы, рожденные в эпоху меркантилизма, повзрослевшие в век пара и стали и угасающие на пороге XXI века, эры информации. Дрюкер утверждает: «Самая нужная вам информация – информация об окружающем мире, и этой информации у вас нет».

В последние несколько лет XX века, по мере раздувания мыльного пузыря доткомов, целые цистерны чернил уходили на рекламу, твердившую, что в Интернете существует «новая экономика». Знатоки утверждали, что продажа вещей в Паутине, где магазины строят из страниц, а не из кирпичей, принципиально отличается от привычных стилей бизнеса и что «старую экономику» уже не оживить. Разумеется, почти все компании новой экономики мертвы, финансисты, поддержавшие их, пережили шок, а эксперты, пропагандировавшие новую экономику, теперь заявляют, что это была пустая мечта. Новая-преновая линия такова, что нам суждено пока оставаться со старой-престарой экономикой.

Вообще говоря, я считаю, что мы живем при новой экономике. Более того, я думаю, что доткомы никогда не были ее частью. Напротив, они стали последним вздохом старой экономики, экономики производства.

В индустриальную эру, до появления программ, продукты создавались из реальных материалов – из атомов. Затраты на добычу, плавку, приобретение, транспортировку, нагрев, формовку, сварку, окраску и снова транспортировку преобладали над всеми прочими расходами. В бухгалтерском учете эти расходы называются «переменными», поскольку они различны для каждого созданного продукта. «Фиксированные расходы», как вы, наверное, догадываетесь, очевидным образом не варьируются и включают такие затраты, как корпоративное администрирование или начальная стоимость завода.

Классические правила управления бизнесом корнями уходят в производственные традиции индустриальной эры. Увы, эти правила не учитывают новые реалии эры информационной, в которой продукты уже не создаются из атомов, а состоят в основном из программного кода, из наборов битов. А биты не подчиняются тем же экономическим правилам, что атомы. Некоторые фундаментальные истины остаются справедливыми и для новой экономики. Цель любого бизнеса – стабильные прибыли, и есть лишь один законный способ получать их: продавать товары или услуги дороже, чем обходится их создание. Из этого следует, что есть два пути повышения прибыльности: снижение затрат или повышение прибылей. В старой экономике лучшим способом было снижение затрат. В новой экономике намного, намного лучше работает повышение прибылей.

Сегодня наиболее нужные и дорогие продукты состоят (полностью или почти полностью) из программного обеспечения. Они не требуют сырья. У них нет стоимости производства. Они не требуют затрат на транспортировку. Не требуют литья, обработки, покраски. Вот она, реальная разница между экономикой индустриальной эры и экономикой эры информационной: в информационную эру отсутствуют или невелики переменные затраты, тогда как в позднюю индустриальную эру именно эти затраты были главным фактором. Действительно, именно отсутствие переменных затрат делает новую экономику новой.

Зарплата программистов в вашем штате – фиксированные затраты или переменные? Один час работы программистов нельзя связать с одной продажей продукта – один и тот же код можно продавать много раз. Вложение в программирование можно амортизировать продажей миллионов копий продукта – точно так же, как продажа продуктов, созданных на заводе, амортизирует вложения в этот завод.

Стоимость создания программ не переменна, но и не фиксирована тоже. Разработка программ – для компании процесс непрерывный, генерирующий прибыли, и это совсем не то же самое, что строительство завода. Высокооплачиваемые строители завода после завершения работ уходят на другую рабочую площадку. Программисты стоят гораздо дороже плотников и сварщиков и никогда не исчезают, потому что, по всей видимости, их работа никогда не завершается. Кто-то может сказать, что программирование – это исследования и разработка, и сходство действительно есть. Однако же исследования и разработка – это размышления и эксперименты, призванные оценить теоретическую жизнеспособность продукта, и они происходят совсем не так, как настоящее создание продуктов. Эта мысль подтверждается тем, что традиционный бухгалтерский учет разделяет исследования/разработку и ежедневную деятельность, приносящую прибыли. Создание программ не вписывается как следует ни в одну из этих категорий учета, приемлемых для прежних предприятий.

Да, этим маленьким терминологическим несоответствием можно было бы и пренебречь как софизмом, уместным в беседе счетоводов за кружкой пива, но в действительности оно оказывает огромное влияние на финансирование разработки программ, на управление проектами по разработке и, что самое важное, на то, как к разработке программ относятся высокопоставленные руководители.

Программисты создают приложения, а руководящие работники создают потоки прибылей и структурные подразделения. Программисты оценивают свой успех по качеству продукта, а руководящие работники – по прибыльности вложений. Эту прибыльность они оценивают на языке математических терминов, позволяющем учитывать фиксированные затраты, переменные издержки, затраты на корпоративное администрирование, исследования и разработку, но, к сожалению, не описывающем подходящие модели для программ и программирования. Бухучет – основной язык бизнеса, и перечисленные категории настолько фундаментальны для всех измерений и коммуникаций в бизнесе, что современные руководители полностью их усвоили. Программирование для них – еще одна статья корпоративных расходов, которую следует причислить к одной из существующих категорий. На практике большинство руководителей расценивают программирование как производственный процесс, имеющий переменные издержки. (Для целей налогообложения в большинстве компаний, создающих программные продукты, программирование проходит по статье исследований и разработок, но во всех остальных отношениях расценивается как деятельность с переменными издержками.) Это худший выбор из всех существующих, поскольку он наносит серьезный ущерб возможности принимать эффективные решения, связанные с бизнесом.

В индустриальную эру ключевым преимуществом была массовость, которая позволяла снижать цены и делать продукцию доступной многим людям. Покупатель при этом получал возможности, ранее не существовавшие или получаемые в результате дорогостоящей ручной работы. Компании соревновались в области продажных цен, непосредственно связанных с

переменными затратами – затратами на производство и доставку. В информационную эру доступность продукции по разумным ценам считается обстоятельством само собой разумеющимся. В конце концов, программы можно распространять через Интернет – практически бесплатно и почти не прилагая усилий.

Бизнес может повышать доходность за счет увеличения прибылей или сокращения издержек. Другими словами, предприятие может увеличивать инвестиции в области фиксированных затрат, повышая качество продукции и укрепляя таким образом ценовые позиции, или же снижать переменные затраты, что означает снижение стоимости производства. В старой, «атомной» экономике снижение издержек давалось легко и было эффективным и предпочтительным. Сегодня же руководители, равняющие программирование с производством, воображают, будто снижение стоимости программирования дается так же легко и оказывается таким же эффективным. К сожалению, старые правила больше не действуют.

Поскольку производство программ сопровождается незначительными переменными издержками, снижение этих издержек не дает преимущества в бизнесе. С точки зрения бухгалтера зарплаты программистов – переменные затраты, однако в действительности их зарплаты представляют собой долгосрочные вложения, фиксированные издержки. Снижение стоимости программирования и снижение стоимости производства – разные вещи. Первое можно сравнить, скорее, с раздачей работникам дешевых инструментов, чем со снижением зарплат. Компании, заказывающие разработку в других странах с целью снижения зарплат, просто не понимают сути дела.

Более того, единственный возможный источник экономического подъема – это повышение качества и, как следствие, привлекательности продукта или услуги, а повышения качества невозможно добиться, сокращая затраты на проектирование и программирование. Правда в том, что в исследования, анализ, планирование и проектирование следует вкладывать больше времени и денег, чтобы полученный результат лучше соответствовал потребностям покупателей.

Разумеется, такой подход требует мышления, с которым не знакомы деловые люди XXI века. Им следует не снижать затраты на создание каждого объекта в отдельности, но повышать затраты на создание всех объектов в совокупности. Это сущность новой экономики, и именно об этом говорил Питер Дрюкер.

Современные фармацевтические компании, работающие над созданием высокотехнологичных лекарств, имеют что-то общее с новой экономикой программного обеспечения. Действительная стоимость производства одной таблетки мизерна, однако разработка лекарства может стоить миллиарды долларов и продолжаться более десяти лет. Подъем после выхода на рынок нового волшебного лекарства может длиться до бесконечности, а вот выпуск недоработанного лекарства способен принести только катастрофический спад. Фармацевтические компании знают, что снижение издержек на разработку – нежизнеспособная стратегия.

Как и разработка лекарств, разработка программного обеспечения совсем не похожа на строительство завода. Завод – физический актив, который принадлежит компании, а работники завода в общем случае легко заменяются. Неосязаемые, но невероятно сложные узоры мыслей, составляющие программное обеспечение, обладают ценностью только в сочетании с написавшим код программистом. Ни одна компания не может себе позволить относиться к программистам так же, как к заводу. Программисты требуют постоянного внимания и поддержки, причем гораздо больших, чем какой бы то ни было завод.

Чаще всего пытаются сэкономить на архитектуре программного продукта, а эта часть проектирования (во время которой изучаются пользователи, определяются сценарии работы, проектируется взаимодействие, определяется форма, описывается поведение) выполняется человеком. Конечно, иногда проектированию уделяют слишком большое внимание, но сокращение этой фазы точно не принесет пользы. Каждый доллар и каждый час, потраченные на архитектуру, обернутся десятикратной экономией на этапе

программирования. Кроме того, вложение в достаточно качественное проектирование делает ваш продукт привлекательным, а это означает, что продукт принесет больше денег. Его привлекательность станет основой для вашего брэнда, расширит возможности для повышения цен, сделает клиентов лояльными, подарит вашему продукту более долгую и насыщенную жизнь. И хотя здесь нет экономии средств, вы получаете большое преимущество в смысле качества. По иронии судьбы лучший способ увеличить прибыльность в информационную эру состоит в том, чтобы больше тратить.

К сожалению, в большинстве руководителей живет практически непобедимое стремление сокращать вложения времени и денег в программирование. Они ошибочно считают устаревшую тактику сокращения издержек подходящей и не понимают, что сокращение инвестиций в программирование оказывает сильное негативное влияние на качество, привлекательность и прибыльность продукта в долгосрочной перспективе. Разумеется, простым повышением затрат не добиться улучшений, а часто ситуация и ухудшается, если дополнительные деньги вливаются в обход мудрости, анализа, правильного руководства. Мой первый наставник, Дэн Хоакин (Dan Joaquin), любил повторять, что правильный обратный вариант старой истины «получаешь то, за что платишь» звучит так: «не получаешь того, за что не платишь». Действия без планирования всегда чреваты риском потратить слишком много. Фокус в том, чтобы потратить нужное количество денег, и он требует значительных познаний в управлении созданием программного обеспечения. Он требует также и процессов, обеспечивающих руководителей пониманием и информацией для принятия верных решений. Дать компаниям такие процессы – цель этой книги.

В буме доткомов участвовали компании с бизнес-моделями, полностью ориентированными на снижение переменных затрат. Хотя многие доткомы рекламировали преимущества покупок через Интернет, их сайты, тяжеловесные и неудобные, выглядели бледно по сравнению с обычной поездкой в торговый комплекс. Основатели доткомов просто лучились от восторга (и пресса, кстати, тоже), потому что им удалось создать предприятия розничной торговли с невероятно низкими переменными затратами. Феерический провал этих предприятий, несомненно, доказывает, что информационной эрой правят иные экономические правила.

В старой экономике более низкие переменные затраты приводили к более широкому распространению товара и снижению розничных цен. Это двойное преимущество было выгодно непосредственно покупателям, а покупатели – фундамент экономического успеха индустриальной революции. В новой экономике успех бизнеса зависит от способности дать потребителям что-то новое и более качественное. Реальное качество каждого шага транзакции – от просмотра страниц до сравнения товаров – должно быть ощутимо более высоким для пользователя. Гораздо приятнее сделать покупку обычным путем, чем продираться через 11 экранов и в конечном итоге выяснить, что все равно придется звонить в компанию. Покупки в сети становятся совершенно ненужными и непривлекательными, если требуется набрать свое имя, свой адрес и ввести информацию о кредитной карте три или четыре раза, а затем обнаружить, что сайт не позволяет купить все необходимое и все равно придется ехать в обычный магазин, сделанный из атомов. Сегодня простое снижение цен на продукты уже не дает гарантии успеха.

Компания Pets.com, специализирующаяся на продаже корма для собак через Интернет, не предлагала более качественный корм, как не предлагала и шоппинг, более приятный, чем в обычном местном магазине; она не предлагала новую информацию, новые возможности, новую уверенность. Она предлагала лишь дешевую доставку, складирование и торговлю (все это переменные затраты) на сайте *Pets.com* . Компания применила классическую тактику снижения затрат, присущую экономике индустриальной эры, полностью игнорируя фундаментальные принципы новой экономики. Конечно, это было еще не первое дыхание новой экономики, но для старой это было последнее издыхание. Я совершенно убежден, что любой товар можно продавать через Интернет успешно и прибыльно. Фокус в том, чтобы в онлайновом магазине было бы ощутимо приятнее покупать, чем в конкурирующих

розничных сетях, и цена здесь – всего лишь одна из составляющих. Есть лишь один способ добиться этого: архитектуру системы следует создавать с целью максимально удовлетворить конечного пользователя. Отношение к любому аспекту проектирования и создания программного обеспечения, как к производственному процессу, служит причиной провала. Проектирование и программирование – попросту неподходящие цели для традиционных методов сокращения издержек. Конечно, можно потратить на создание программ слишком много времени и денег, но опасность потратить меньше необходимого гораздо серьезнее.

Скорее всего, эта опасность вам знакома и удивления не вызывает, но для большинства высокопоставленных руководителей крупных компаний немыслима сама идея. Эти руководители до сих пор работают по моделям бухгалтерского учета, вошедшим в моду в эпоху паровых машин, тогда как все аспекты жизни их компаний – функционирование, принятие решений, коммуникации и финансы – полностью зависимы от программного обеспечения. Термины и понятия, которыми оперируют эти руководители, просто не принимают во внимание уникальную природу бизнеса в эпоху, когда инструменты и продукты коммерческой деятельности являют собой неосязаемые переплетения битов вместо вагонов с железом. Отмечу, впрочем, что куклы из носков – идея классная.

Корпорации уже нанимают проектировщиков взаимодействия и начали применять целеориентированный подход, однако качество программных продуктов не слишком улучшилось. Более того, высокие затраты на программирование и неподатливость процесса создания программ остаются на своих местах. Почему?

Перемены невозможны, пока администраторы компаний не осознают, что проблемы с программами – это не технические сложности, а значимые для бизнеса вопросы. Наши проблемы останутся неразрешенными до тех пор, пока мы не изменим процесс и организацию.

Компании живут не только по устаревшим финансовым моделям, но еще и по негодной организационной модели. Эта модель – копия той, что имеет хождение в учебных заведениях, в ней создание программы смешивается с планированием и решением инженерных задач. Такова природа исследований. Прискорбно, что эта парадигма без объявления войны была перенесена в неизменном виде в мир бизнеса, где ей не место.

Корни всех современных производственных дисциплин уходят в доиндустриальные времена. Всех, кроме дисциплины программного обеспечения, которая возникла, когда индустриализация уже закончилась. Только программирование произошло сразу из учебных заведений, где время исследований не ограничивалось, студенческой рабочей силы было хоть отбавляй, о прибылях вообще не говорили, а неработающая программа могла сойти за весьма удачный эксперимент. Неслучаен тот факт, что Microsoft, IBM, Oracle и другие ведущие компании-производители ПО расположены в «кампусах». Университетам не нужны прибыли, они не стараются успеть вовремя создать привлекательные и полезные продукты.

Любой бизнес, не связанный с программным обеспечением, начинается с исследований и заканчивается распространением продуктов или предложением услуг. Компания тщательно планирует время между этими двумя событиями, осознавая, что преждевременный выпуск непродуманного продукта опасен как для банковского счета, так и для ее репутации. Руководители знают, что время, размышления и деньги, вложенные в планирование, обернутся крупными дивидендами – гладким и быстрым процессом производства, популярностью и прибыльностью конечных продуктов.

Во всех других конструкторских дисциплинах инженеры создают стратегию, а ремесленники претворяют ее в жизнь. Инженеры не занимаются строительством мостов, этим занимаются другие специалисты. Только в области программного обеспечения перед инженером ставится задача создать собственно продукт. Только в области программного обеспечения перед «строителем мостов» ставится задача определить, как следует создавать продукт. Только в области программного обеспечения эти две задачи решаются не последовательно, а одновременно. Однако компании, создающие программы, похоже, не осознают существования такой аномалии. Инженерное дело и конструкторское дело так

плотно пересекаются, что специалисты и руководители их не разделяют и, вероятно, не различают. Планированием любого рода здесь пренебрегают или откладывают его до тех пор, пока не станет слишком поздно. Считается нормальным откладывать решение очень сложных инженерных проблем до момента, когда уже экономически слишком накладно откатываться к фазе проектирования, потому что полным ходом пишется код для коммерческой версии продукта.

Проектирование архитектуры следует начинать на ранних стадиях инженерного планирования. Более того, именно оно должно быть движущей силой на этих стадиях, но такие разработки обычно откладываются до момента старта проекта и ведутся параллельно с созданием кода, поэтому не занимают должного места в процессе конструирования продукта. Компании нанимают проектировщиков взаимодействия и обучают эргономистов создавать персонажей, однако работа этих людей почти не влияет на стоимость разработки и качество завершенного продукта.

Решение проблемы – в руках президентов и исполнительных директоров корпораций. Делегируя решение своим техническим директорам или вице-президентам разработки, они поступают неверно. Эти достойные исполнители – технари, а проблема не имеет отношения к технике. Как сказал Дрюкер, инструменты для бухгалтерии, на которые полагаются директора компаний, попросту не отражают истинной природы этих компаний. Нельзя ведь на основе точных показаний спидометра утверждать, что автомобиль движется в нужном направлении. В мире бизнеса цифровых технологий такой подход не может быть эффективным.

Одна из серьезных проблем применения неверных методов бухучета и организации для разработки программ состоит в том, что руководители не осознают, во что обходится каждый доллар затрат на программирование. Точная система показала бы, что из каждого доллара около пятидесяти центов тратится неправильно и что еще два или три доллара требуется, чтобы исправить проблему, вызванную этим некорректным вложением средств. В любом другом бизнесе подобная статистика вызвала бы революцию, однако отрасль программного обеспечения продолжает жить в состоянии блаженного неведения.

За последние тринадцать лет фирма Cooper выступала в роли консультанта сотен компаний. Мои талантливые проектировщики создали для большинства клиентов «чертежи» продуктов, позволяющие коренным образом изменить ситуацию, но лишь немногие сумели воспользоваться всеми полученными преимуществами. В большинстве этих компаний проектирование взаимодействия и архитектуру программ считают лишь советом, в этих компаниях последнее слово всегда остается за программистами и инженерами. Ни один из президентов в этих компаниях не имеет ни малейшего понятия о том, что происходит в боксах инженеров, и потому расписание ужимается безо всякой причины. Программисты постоянно работают в условиях дефицита ресурсов и, прежде всего – времени на хорошее программирование, а также времени, чтобы определить, где вообще требуется программирование. Они вынуждены защищаться, отвергая советы, и изворачиваться, общаясь со своими менеджерами.

На мой взгляд, существует два типа руководителей: инженеры и запуганные инженерами. Первые множат знакомые проблемы, поскольку их точка зрения безнадежно испорчена конфликтом интересов. Вторые множат проблемы, поскольку не умеют говорить на языке программистов. И я не имею в виду языки Java и C#. Я имею в виду, что у деловых людей и программистов нет общих инструментов и общих целей. Человек разумный делегирует человеческие проблемы хомо логикус, не осознавая, что решение могло бы оказаться намного более приятным в случае применения – на исполнительном уровне – уместных финансовых и организационных моделей.

У компаний есть отличная возможность сдвинуться с мертвой точки и сосредоточить усилия на удовлетворении потребностей клиентов, а не на программах, на персонажах, а не на технологиях, на выгоде, а не на программистах. Я с нетерпением ожидаю появления просвещенного руководителя, который ухватится за эту возможность и навсегда изменит

способ создания программного обеспечения, подав смелый и успешный пример.

Алан Купер,
Пало, Альто, Калифорния
www.cooper.com
inmates@cooper.com

Часть I
Компьютерная безграмотность

Глава 1
Загадки века информации

Что получится, если скрестить компьютер с самолетом?

В декабре 1995 года рейс 965 компании American Airlines вылетел по регулярному маршруту из Майами в Кали, Колумбия. На подлете к посадочной полосе пилоту Боинга-757 потребовалось выбрать следующий радиомаяк по имени «ROZO». Он набрал букву «R» в своем навигационном компьютере. Компьютер отобразил перечень ближайших радиомаяков с именами на «R», а пилот выбрал первую позицию в списке, потому что широта и долгота показались ему верными. К несчастью, вместо «ROZO» пилот выбрал маяк «ROMEO», расположенный в 210 километрах к северо-востоку. Самолет направлялся на юг и находился в тот момент в долине, пролегающей с юга на север, так что любое отклонение от курса было опасно. Следуя показаниям полетного компьютера, пилоты начали корректировать курс к востоку, и самолет врезался в гранитный пик на высоте трех километров. Сто пятьдесят два пассажира и восемь членов экипажа погибли. Четыре пассажира выжили, получив серьезные травмы. Национальная комиссия по безопасности транспорта провела расследование и – как обычно – заявила, что причиной явился человеческий фактор. Вспомогательное навигационное средство, показаниями которого руководствовались пилоты, выдало корректную информацию, но не для посадки в Кали. Человеческий фактор, если следовать буквальному смыслу фразы, действительно был причиной – ведь именно пилот выбрал неправильный маяк. Однако если взглянуть на ситуацию в целом, вины пилота здесь не было.

Передняя панель навигационного компьютера самолета отображала выбранный навигационный маяк и индикатор отклонения от курса. Когда самолет находится на курсе, стрелка расположена по центру, но она никаким образом не указывает на правильность выбора радиомаяка. Индикатор выглядит примерно одинаково перед посадкой и перед катастрофой. Компьютер сообщил пилоту, что на выбранный маяк взят точный курс. К сожалению, компьютер упустил из виду, что такой выбор маяка смертелен.

* * *

Полученная информация может быть точной и полной, но при этом трагически некорректной. Это происходит слишком уж часто, когда мы общаемся с компьютерами, а компьютеры проникли во все аспекты современной жизни. От самолетов, на которых мы летаем, до потребительских товаров и услуг – везде компьютеры, везде присущее им поведение и способы взаимодействия.

В компьютерной индустрии широкое хождение имеет такой анекдот: человек, пилотирующий небольшой самолет, заблудился в облаках. Он снижается и замечает офисное здание неподалеку. «Не подскажете, где я нахожусь?» – кричит он человеку в открытом окне. Человек отвечает: «Вы в самолете, примерно в тридцати метрах над землей». Пилот немедленно ложится на верный курс, находит аэропорт и совершает посадку. Его пассажиры

в изумлении интересуются, как он определил, куда лететь. И пилот говорит: «Ответ этого человека был абсолютно точен и правдив, однако совершенно бесполезен, поэтому я сразу понял, что это разработчик программного обеспечения из Microsoft, а я знаю, где находится здание Microsoft по отношению к аэропорту».

В свете трагедии рейса 965 анекдот звучит зловеще, однако профессионалы из цифрового мира рассказывают его часто и с удовольствием, потому что он отражает главную правду о компьютерах: они могут сообщать нам факты, но не информируют нас. Их указания точны, но не способны привести нас в нужное место. Полетный компьютер рейса 965 мог с легкостью сообщить пилотам, что «ROMEO» – неподходящий маяк для Кали. Даже простой намек, что выбор «необычен» или «незнаком» мог бы спасти самолет. Вместо этого компьютер, похоже, совершенно не интересовали пассажиры и собственно рейс. Его интересовали только собственные вычисления.

Сложные в применении компьютеры влияют на всех нас, временами фатально. Продукты, основанные на программном обеспечении, сложны в применении не от природы, но потому, что мы используем неверный процесс для их создания. В данной книге я намереваюсь показать следствия этого неверного процесса и объяснить его происхождение. Затем мы поговорим о том, как следует изменить процесс, чтобы наши программные продукты стали дружелюбными, мощными и приятными. В этой главе я прежде всего покажу, насколько серьезна эта проблема.

Что получится, если скрестить компьютер с фотокамерой?

Вот загадка информационного века: что получится, если скрестить компьютер с фотокамерой? Ответ: компьютер! Тридцать лет назад в моем первом фотоаппарате, 35-миллиметровом Pentax H, была маленькая батарейка, питавшая экспонометр. Я просто менял батарейку каждые два года, как в наручных часах.

Пятнадцать лет назад в моей первой электронной фотокамере, 35-миллиметровом Canon T70, было две пальчиковых батарейки, приводивших в действие достаточно простой компьютерный блок экспонометра и питавших автоматическую прокрутку пленки. Простой выключатель на аппарате предотвращал ненужные затраты энергии батареек.

Пять лет назад в моем Logitech, цифровом фотоаппарате первого поколения, тоже был подобный выключатель, однако на этот раз в камере уже появились зачатки компьютерных мозгов. Так что если я забывал выключить ее, она автоматически выключалась через минуту бездействия. Симпатично.

Год назад моя цифровая фотокамера второго поколения, Panasonic PalmCam, содержала еще более сообразительную компьютерную микросхему. Настолько сообразительную, что простой выключатель эволюционировал в переключатель Off/Rec/Play. Появились режимы: чтобы снимать, необходимо было перевести камеру в режим Rec, а чтобы просматривать фотографии на маленьком экране – в режим Play.

Моя последняя фотокамера – Nikon CoolPix 900 – цифровой фотоаппарат третьего поколения, и она еще умнее. Настолько умнее, что содержит полноценный компьютер, отображающий песочные часы а-ля Windows при «загрузке». Словно какая-то рыба-мутант с лишними головами, выключатель дорос уже до *четырех* позиций: Off/ARec/MRec/Play. ARec обозначает автоматическую запись, а MRec – ручную. Насколько я могу судить, разницы между этими режимами нет. Режим On (включено) вообще отсутствует, и без подробных объяснений никто из моих друзей не может сообразить, как включить устройство.

Эта новая камера весьма прожорлива, поэтому создатели заботливо снабдили ее изощренной компьютерной программой, управляющей потреблением энергии батарей. Типичный сценарий выглядит так: я перевожу зловещий переключатель в положение MRec, жду примерно семь длинных секунд, пока камера загрузится, затем направляю ее на предмет съемки. Я нацеливаю камеру и выбираю увеличение, чтобы получить нужный кадр. В тот момент, когда я уже почти нажимаю спуск, камера внезапно осознает, что одновременная зарядка вспышки, увеличение и включение экрана окончательно исчерпали заряд аккумулятора. В порыве самозащиты камера временно отключает возможность снимать. Но я этого не знаю, потому что смотрю через видоискатель, машу руками, говорю «улыбочку» и нажимаю на спуск. Компьютер фиксирует нажатие на кнопку, но не способен повиноваться. В ложном порыве спасения программа управления питанием мгновенно вмешивается и принимает ответственное решение: снизить нагрузку. Она отключает прожорливый LCD-экран. Я в недоумении смотрю на камеру, силясь понять, почему она не сделала снимок, пожимаю плечами и опускаю руку с камерой. Но после отключения экрана другие подсистемы получают больше энергии батареи. Программа управления питанием ощущает повышение потенциала и осознает, что *вот теперь* электричества достаточно, чтобы сделать снимок. Она возвращает управление основной программе, которая терпеливо ожидает, когда можно будет выполнить мою команду сделать снимок, и делает замечательный цифровой снимок моего колена с автофокусом, экспозицией и с высоким разрешением.

В моем старом механическом Pentax была ручная фокусировка, ручная экспозиция, ручная выдержка, однако пользоваться им было гораздо менее мучительно, чем полностью компьютеризованным современным Nikon CoolPix 900, в котором фокусировка, экспозиция и выдержка автоматические. Эта фотокамера по-прежнему позволяет фотографировать, но ведет себя уже не как фотокамера, а как компьютер.

*** * ***

Лягушка, попавшая в кастрюлю с холодной водой на плите, не осознает, насколько убийственно повышение температуры. Напротив, тепло притупляет чувства лягушки. Подобно лягушке, я не осознавал медленного марша моих фотокамер от простого к сложному по мере их компьютеризации. Все мы переживаем точно такое же, медленное, анестезирующее вторжение компьютерного поведения в нашу повседневную жизнь.

Что получится, если скрестить компьютер с будильником?

Компьютер! Я только что купил для спальни новые дорогие часы со встроенным радиоприемником – JVC FS-2000. Прибор оснащен весьма изощренным компьютерным мозгом, высокоточными часами, цифровым звуком и вообще множеством функций. Он будит меня в заданное время, проигрывая музыку с компакт-диска, и обладает достаточно деликатным характером и достаточной сообразительностью, чтобы мееееедленно прибавлять звук, если дело происходит в шесть утра. Весьма приятная и довольно редкая особенность, компенсирующая мое желание вышвырнуть эту возмутительную машину из окна.

Очень сложно определить, когда будильник включен, поэтому время от времени он пропускает побудку в понедельник и выдергивает меня из кровати рано утром в субботу. Разумеется, в этих часах есть индикатор активности будильника, однако это не означает, что его можно использовать. Встроенный сложный алфавитно-цифровой жидкокристаллический дисплей (LCD) отображает всевозможные функции часов. Так, маленькое изображение часов в левом верхнем углу дисплея указывает, что будильник включен, однако в полутьме спальни этот символ разглядеть невозможно. Дисплей оснащен подсветкой, благодаря которой символ часов становится видимым, однако подсветка включается, лишь если самостоятельно включить радио или проигрывание компакт-диска. Но тут есть одна

тонкость – будильник (даже активизированный) ни за что не сработает, если оставить проигрыватель компакт-дисков включенным. Именно такое парадоксальное поведение часто застает меня врасплох.

Отключить будильник просто: достаточно нажать кнопку Alarm один раз, и часики исчезнут с дисплея. Но чтобы включить будильник, я должен нажать эту кнопку ровно пять раз. После первого нажатия дисплей отображает время, когда сработает будильник. После второго – время, когда будильник перестанет работать. После третьего – источник звука, радио или компакт-диск. После четвертого – предустановленный уровень звука. После пятого часы возвращаются в нормальный режим работы с включенным будильником. Однако всего одно дополнительное нажатие отключает будильник. В полусне, в темной спальне достаточно сложно правильно исполнить этот маленький цифровой балет.

Будучи занудным поклонником всяких штуковин, я продолжаю свои игры с прибором в надежде справиться с ним. А вот моя жена давно уже сдалась на милость дьявольской машины. Ей нравится гладкий современный дизайн и качество звука, но прибор не прошел аттестацию на часы-будильник, потому что его слишком сложно заставить работать. Этот будильник по-прежнему способен разбудить меня, но ведет себя, словно компьютер.

А мой старый некомпьютерный будильник за 11 долларов будил меня внезапным злобным жужжанием. Когда будильник был включен, горела красная лампочка. Когда он был выключен, лампочка не горела. По многим причинам этот старый будильник мне не нравился, однако я, по крайней мере, мог определить, собирается ли он меня будить.

* * *

Производителям гораздо дешевле использовать компьютеры для управления внутренними процессами устройств, чем более старые, механические методы, а потому проникновение компьютеров во все продукты и услуги в нашей жизни экономически неизбежно. Это означает, что поведение всех существующих продуктов скоро станет похожим на поведение самых отвратительных компьютеров, если только мы не применим другой подход.

* * *

Описанное явление не ограничено лишь продуктами для конечного потребителя. Прочти любое компьютеризованное устройство или услуга предлагают больше возможностей и вариантов использования, чем механический эквивалент. Однако на практике именно механическими устройствами мы пользуемся более гибко, более тонко, с большим пониманием, чем современными вариантами тех же устройств, основанными на кремниевых микросхемах.

Высокотехнологические компании, стремясь улучшить свои продукты, просто насыщают их сложными и никому не нужными возможностями. Поскольку ущербный процесс не способен решить проблему некачественных продуктов, а позволяет лишь добавлять новые функции, именно этим создатели продуктов и занимаются. Позже в этой книге я покажу, как усовершенствованный процесс разработки делает пользователей счастливыми, не требуя затрат времени на дополнительные ненужные функции.

Что получится, если скрестить компьютер с автомобилем?

Компьютер! Великолепный новый высокотехнологичный спорткар Boxster от Porsche оборудован семью компьютерами, которые управляют его сложными системами. Один из них занимается исключительно двигателем. В этот компьютер встроены специальные процедуры, позволяющие выходить из критических ситуаций. К сожалению, эти процедуры иногда становятся причиной странных эффектов. В некоторых ранних моделях, если уровень топлива в баке становился очень низким – около четырех литров – центробежная сила в крутом повороте могла сместить бензин к одной стенке бака, из-за чего воздух попадал в топливную систему. Компьютер фиксировал серьезное изменение в поступающей топливной смеси и интерпретировал это как катастрофический сбой системы впрыска. Чтобы избежать повреждений, компьютер отключал зажигание и останавливал автомобиль. Кроме того, чтобы избежать повреждений, компьютер не разрешал водителю перезапускать двигатель, пока машину не отбуксируют в мастерскую и не починят.

Когда владельцы первых Boxster столкнулись с этой проблемой, компания Porsche смогла предложить им только одно решение: открыть капот и отсоединить батарею как минимум на пять минут, чтобы компьютер смог забыть о заминке. Эти спортивные машины позволяют носиться по двухполосным асфальтовым дорогам, но в острых поворотах теперь ведут себя точно как компьютер.

* * *

Прилагая достойные всяческих похвал усилия, чтобы обезопасить владельцев Boxster, программисты превратили этих владельцев в униженных жертв. Каждый приверженец спортивных автомобилей знает, что компания Porsche не скупится на уважение к своим клиентам и предоставляет им множество привилегий. Этот инцидент показывает, что программное обеспечение автомобиля создано не фирмой Porsche, сделавшей другие компоненты автомобиля. Оно создано компанией внутри компании: программистами, а не легендарными немецкими автостроителями. Каким-то образом появление новой технологии вынудило солидную компанию оступиться и упустить из виду некоторые из своих ключевых принципов. Стандарты качества у разработчиков программного обеспечения гораздо ниже, чем в традиционных инженерных дисциплинах.

Что получится, если скрестить компьютер с банком?

Компьютер! Всякий раз, снимая деньги в банкомате, я сталкиваюсь с одним и тем же угрюмым и сложным поведением, столь присущим компьютерам. Если я сделаю малейшую ошибку, банкомат блокирует всю транзакцию и вышвырнет меня из процесса. Я должен вытащить карту, снова вставить ее, повторно набрать свой PIN-код, а затем повторить запрос. Обычно и ошибка-то не моя, это компьютер банкомата дипломатично сбивает меня с толку. Он постоянно спрашивает, с какого счета я хочу снять деньги – с текущего, с депозитного или же с валютного – хотя счет у меня всего один, текущий. Я постоянно забываю, какого типа у меня счет, и такой вопрос сбивает меня с толку. Примерно раз в месяц я безо всякого умысла выбираю депозитный счет, и адская машина бесцеремонно заставляет меня начать все сначала. Чтобы отказать в снятии денег с депозитного счета, машина должна знать, что у меня такого счета нет, но она предлагает мне этот счет в качестве одного из вариантов. Единственная разница между мной, когда я выбираю банковский счет, и пилотом рейса 965, выбравшим «ROMEO», – в масштабах последствий.

Кроме того, банкомат налагает «суточное ограничение» в размере двухсот долларов. Если я совершу все шаги – наберу свой код, выберу тип счета, укажу сумму – и это будет, скажем, сумма в 220 долларов, компьютер бесцеремонно откажет в проведении операции, грубо проинформировав меня, что я превысил суточное ограничение. Он не сообщит мне, каково это ограничение, и не позволит узнать, сколько денег на моем счету, и не даст возможности указать другую, меньшую, сумму. Он просто выплевывает мою карту и

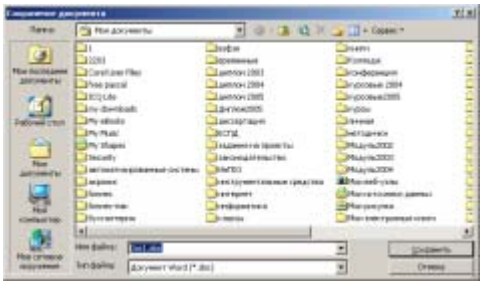
предоставляет мне возможность повторить процесс с самого начала, при том, что я обладаю ровно той же информацией, что и минуту назад, а очередь за мной растет, волнуется и вздыхает. Банкомат точен и правдив, но совершенно бесполезен.

Этот банкомат следует заложенным в него правилам, и я тоже вполне готов им следовать, но это уж слишком компьютерный подход – не сообщить мне о правилах, дать мне противоречивые сведения, а затем бесцеремонно наказать меня за то, что я эти правила по незнанию нарушил. Такое поведение – столь типичное для компьютеров – не присуще им от природы. По сути дела, от природы компьютерам ничего не присуще: они просто действуют, повинуясь программе. А программы пластичны так же, как человеческая речь. Человек может говорить грубо или вежливо, угрюмо или любезно. Так же легко, как человек способен вежливо говорить, компьютер может уважительно и с почтением себя вести. Требуется лишь, чтобы кто-нибудь объяснил, каким образом. К сожалению, программисты не слишком успешно учат компьютеры подобным вещам.

Компьютер позволяет легко попасть в беду

Компьютеры на рабочих столах ведут себя тем самым вызывающим раздражение способом, который им так присущ; им даже не требуется какое-либо скрещивание. Моя подруга Джейн когда-то работала координатором в области связей с общественностью. Она работала в Microsoft Word на своем компьютере под управлением Windows 95 – редактировала записки и контракты. Файловая система в Windows 95 имеет иерархическую структуру. Все документы Джейн хранились в маленьких папках, которые хранились в других маленьких папках. Джейн этого не понимала, равно, как не видела преимуществ в таком хранении информации. Вообще говоря, Джейн не слишком над этим задумывалась, а просто следовала по пути наименьшего сопротивления.

Джейн только что вчерне закончила новый контракт на пиар для начинающей компании из Кремниевой Долины. Она выбрала Закрыть из меню Файл. Вместо того чтобы сделать как сказано и закрыть документ, программа Word открыла диалог. Разумеется, речь идет о до боли знакомом запросе Do you want to save changes? (Сохранить изменения?). Она ответила – как обычно – нажатием кнопки Yes. Она так часто давала этот ответ, что даже перестала смотреть на диалоговое окно.



За первым диалогом немедленно последовал еще один – тоже очень знакомый Save As (Сохранить как). Этот диалог явил Джейн множество непонятных кнопок, пиктограмм и текстовых полей. Единственное, что Джейн здесь понимала и чем пользовалась, – поле ввода для имени файла. Она набрала подходящее имя и нажала кнопку Сохранить. Программа сохранила контракт в папке *Мои документы* . Джейн настолько привыкла к этой ненужной процедуре, что проходила ее, не задумываясь.

В обед, пока Джейн не было в офисе, Сунил, компьютерный специалист компании, установил на ее машину новую версию антивируса VirusKiller. Работая на компьютере Джейн, Сунил воспользовался программой Word, чтобы просмотреть файл Readme для VirusKiller. Просмотрев файл, Сунил закрыл его и привел компьютер Джейн в прежнее состояние. То есть он так думал.

После обеда Джейн понадобилось вновь открыть контракт, чтобы распечатать его и показать шефу. Джейн выбрала Открыть из меню Файл, и появилось диалоговое окно Открыть. Джейн ожидала, что здесь будут выведены в удобном алфавитном порядке все ее контракты и документы. Вместо этого она увидела кучу имен файлов, которые никогда не видела раньше и не могла опознать. Один из этих файлов назывался *Readme.doc* .

Разумеется, когда Сунил использовал Word для просмотра файла Readme, он велел программе заглянуть в загадочную папку на шестом уровне вложенности файловой системы и безо всякого умысла сбил привычную для Джейн настройку на *Мои документы* .

Джейн была озадачена. Первая и неизбежная мысль была о том, что весь ее тяжелый труд каким-то образом оказался стертым, поэтому она не на шутку разволновалась и позвала Рене, свою подругу и коллегу, однако Рене была сбита с толку не меньше Джейн. Наконец, в состоянии близком к панике, Джейн позвонила Сунилу, чтобы попросить о помощи. Сунила на месте не было, и только в понедельник утром он смог заглянуть к Джейн и все исправить. Джейн, Рене, Сунил и компания, в которой они работали, потеряли каждый по половине дня.

Иерархические файловые системы нужны операционным системам компьютеров, но не людям. Неудивительно, что программистам нравится видеть иерархические файловые системы, но точно так же нет ничего примечательного в том, что обычные пользователи, вроде Джейн, никакого удовольствия от этого не испытывают. Вернее говоря, нет ничего примечательного в этом для всех, кроме программистов, создающих для нас программное обеспечение. Они программируют поведение и отображение информации так, что это подходит для них самих, но это очень сильно отличается от того, что подходит для Джейн. За срыв планов и низкую эффективность работы винят Джейн, а не программистов, которые ее подставили.

У Джейн, по крайней мере, есть работа. Ведь многих людей считают недостаточно «компьютерно образованными», а потому не подлежащими найму. Все больше и больше позиций требуют взаимодействия с компьютерами, так что пересекать границу между наймом и отсутствием такового становится все труднее. Политики могут требовать создания рабочих мест для неимущих, однако ни одна компания не позволит недостаточно компетентному человеку сесть за компьютер. Необходимо слишком серьезное обучение, и здесь слишком велик риск разрушения информации и порчи бесценных баз данных.

Возмутительное поведение и невразумительность взаимодействий, присущие продуктам, основанным на программном обеспечении, наделяют законным статусом режим, который я называю «апартеидом программного обеспечения». Этот режим не позволяет нормальным в целом людям выходить на рынок труда и жить в обществе, потому что они не могут эффективно использовать компьютеры. В нашем просвещенном обществе социальные активисты неустанно трудятся, чтобы разрушить расовые и классовые барьеры, в то время как технологи тяжким трудом воздвигают новые, еще более высокие барьеры. Целенаправленно проектируя наши программные продукты так, чтобы они были более гуманными и терпимыми к ошибкам людей, мы автоматически сможем сделать их менее восприимчивыми к социальному классу и цвету кожи.

Коммерческое программное обеспечение тоже страдает

Компьютеры захватывают в авиалайнерах не только кабины пилотов, но и пассажирские салоны, и ведут себя там столь же знакомо извращенным и сложным для восприятия способом. Современные реактивные самолеты оборудованы развлекательными системами, позволяющими пассажирам слушать в полете музыку и смотреть фильмы. Эти

системы – обычные компьютеры, объединяемые локальными сетями, точно как в вашем офисе. Хорошие развлекательные системы устанавливаются обычно только на крупных самолетах, летающих трансокеанскими рейсами.

Развлекательная система одной авиакомпании оказалась столь неприятной в использовании, что многие стюардессы и стюарды пытались добиться перевода на более короткие местные рейсы, чтобы избежать необходимости изучать и использовать эту сложную вещь. Это примечательно, учитывая, что освященный временем процесс служебного роста на авиалиниях основан на старшинстве, так что именно эти дальние маршруты всегда считались наиболее лакомыми кусочками – благодаря продолжительным остановкам в экзотических местах вроде Сингапура и Парижа. Желание стюардов перевестись на непримечательную, неромантическую дерготню рейсов из Денвера в Даллас или Лос-Анжелеса в Сан-Франциско, лишь бы избежать общения с полетной развлекательной системой, свидетельствовало о серьезной проблеме с боевым духом. Любая компания, мучая плохим оборудованием своих наиболее ценных сотрудников, – именно тех, что провели больше всего времени с клиентами, – поступает глупо, расточительно тратя деньги, подрывая преданность клиентуры и собственного персонала.

Компьютерный комплекс другой крупной авиакомпании был еще хуже. Эта авиакомпания создала полетную развлекательную систему, связавшую показ фильма с функцией сбора денег. Раньше, в закрытом реактивном самолете, на высоте одиннадцати километров процедура сбора денег основывалась на принципах доверия: никто ведь не сбежит из самолета на такой высоте. Стюарды обеспечивали пассажиров товарами и услугами, когда это было удобно, а сбор платы был весьма слабо связан с этим процессом, и сотрудникам авиакомпании не приходилось без нужды бегать взад-вперед по узким проходам. Конечно же, время от времени случались ошибки, но их стоимость никогда не превышала нескольких долларов, а система в целом была достаточно человечной и способной прощать ошибки; все были довольны, и работа была не в тягость.

После компьютеризации процесса приема денег за услугу, стюарду пришлось сначала получать плату от пассажиров, затем идти в самое начало салона, где находится консоль управления, набирать пароль обслуживающего персонала и выполнять транзакцию по регистрации оплаты. Лишь когда транзакция завершится, пассажир сможет посмотреть фильм или послушать музыку. Столь бестолковое проектирование продукта заставляло стюардов сотни раз впустую ходить взад-вперед по узким проходам – в течение одного рейса. От отчаяния стюарды устраивали короткое замыкание в начале каждого длинного рейса вскоре после взлета. Затем они вежливо объявляли пассажирам, что, к сожалению, система неисправна и в этом рейсе фильмов не будет.

Авиакомпания потратила миллионы долларов на создание системы столь отвратительной, что пользователи преднамеренно отключали ее, лишь бы не иметь с ней дела. Тысячи скучающих пассажиров – просто невинные жертвы. И все это на долгих трансокеанских рейсах, забитых, как правило, бесценными постоянными клиентами. Не могу назвать точную цифру потерь авиакомпании, но убежден, что все это обошлось ей катастрофически дорого.

Программное обеспечение полетных развлекательных систем работало с безупречной точностью, но не умело взаимодействовать с людьми, и поэтому его внедрение закончилось полным провалом. Почему компания не могла предвидеть столь печальный исход? Почему не увидела эту связь? Цель данной книги – ответить на такие вопросы и показать, как избежать подобных высокотехнологических катастроф.

Что получится, если скрестить компьютер с военным кораблем?

В сентябре 1997 года, участвуя в морских маневрах в Атлантике, корабль ВМФ США *Yorktown* , один из новых крейсеров с оборонительной системой Aegis,[2] замер на месте.

[2] Aegis (англ. *эгида*) интегрирует системы обнаружения и боевые системы корабля с целью противостояния

Техник ВМФ, калибруя топливный клапан, ввел нулевое значение в один из управляющих компьютеров – с процессором Pentium Pro и операционной системой Windows NT. Программа попыталась разделить другое число на этот нуль, то есть выполнить операцию, не определенную в математике, что и стало причиной сбоя всей системы управления бортом. Без участия компьютеров двигатель прекратил работать, и корабль два часа сорок пять минут качался на волнах, пока не прибыл буксир. Хорошо, что это произошло не в зоне боевых действий.

Что получится, если скрестить компьютер с военным кораблем? Адмирал Нимиц[3] в гробу перевернулся бы! Несмотря на описанную неудачу, в ВМФ приняли решение о компьютеризации всех кораблей, поскольку это позволяло сэкономить на персонале, а чтобы отразить критику, объявили причиной «происшествия» человеческий фактор. Раз процесс создания программного обеспечения вышел из-под контроля, индустрия высоких технологий должна либо привести этот процесс в порядок, либо продолжать сваливать вину на обычных пользователей, в то время как все более грандиозные механизмы беспомощно плещутся в воде.

Техноярость

В недавнем номере *WallStreetJournal* появилась статья, посвященная анонимному видеоклипу, широко распространившемуся посредством электронной почты. В клипе «…Усатый Рядовой Гражданин в рубашке с коротким рукавом озадаченно склонился над компьютером. Внезапно, в порыве раздражения, он ударяет по своему монитору. Любопытствующий коллега заглядывает в его отсек, в то время как этот человек лупит клавиатурой по монитору, сшибая его на пол. Поднявшись со своего места, он добивает упавший монитор последним жестоким ударом».

В статье говорилось, что реакцию этот клип вызвал «значительную» и что он, очевидно, затронул «мощную скрытую тенденцию к техноярости».

По иронии судьбы, чтобы даже просто переслать этот видеоклип, необходима умеренная подготовка в компьютерной области. Человек на видео может быть и актером, но он затронул знакомую бизнес-миру струнку сочувствия. В нашей жизни раздраженность сложными и неприятными продуктами, основанными на программном обеспечении, быстро растет.

В закрытых списках рассылки имеют хождение шутки о «компьютерном синдроме Туретта». Это вариация на тему психического расстройства, известного как синдром

ракетным атакам. – *Прим. перев.*

[3] Честер Нимиц, командующий Тихоокеанским флотом США во время Второй мировой войны. – *Прим. перев.*

Туретта. Некоторые из подверженных этому расстройству людей переживают неконтролируемые припадки сквернословия. Шутка в том, что можно пройти по коридорам практически любого современного офисного здания и услышать, как в целом нормальные люди, сидя за своими компьютерами и сжав зубы, постоянно и яростно ругаются. Кто знает, что вызвало такую вспышку: потерянный файл, недоступное изображение или же раздражающее взаимодействие. А может быть, программа просто вежливо стерла единственную копию пятисотстраничной рукописи пользователя, потому что он ответил «Да» на диалог с подтверждением, предположив, что ему предлагается сохранить изменения, когда в действительности предлагалось удалить работу.

Индустрия в «несознанке»

Наш мир опьянен высокотехнологичными инструментами. Компьютеры господствуют на рабочих местах и у нас дома, транспортные средства заполняются примочками, основанными на кремниевой технологии. Каждое из этих мощных, изощренных компьютеризованных устройств обескураживающие сложно и нелогично в применении.

Индустрия высоких технологий отказывается признать простой факт, очевидный каждому владельцу мобильного телефона или текстового редактора: *наши компьютеризованные инструменты слишком сложно применять* . Инженеры, создающие программное обеспечение и высокотехнологичные устройства, довольны собственными усилиями. Разработчики программного обеспечения[4] пытаются в меру возможностей сделать эти инструменты простыми в применении и немного в этом преуспели. Они полагают, что их продукты настолько просты в применении, насколько это технически возможно. Будучи инженерами, они доверяют технологии и верят в то, что лишь новая технология – скажем, распознавание голоса или искусственный интеллект – способна улучшить опыт для конечных пользователей.

По иронии судьбы, вероятно, что *наименьший* вклад в простоту использования продуктов, основанных на программном обеспечении, внесет именно новая технология. *Технически* разницы между сложной, запутанной программой и простым, приятным, мощным продуктом практически нет. Вопрос скорее в культуре, подготовке, отношении людей, создающих эти продукты, нежели в микросхемах и языках программирования. Ущербен наш *процесс* разработки, а не инструменты.

Индустрия высоких технологий по недосмотру поставила во главу программистов и инженеров, поэтому доминирует их сложная в применении инженерная культура. Несмотря на кажущиеся полномочия, люди на руководящих постах попросту не контролируют индустрию высоких технологий. Этим шоу заправляют инженеры. В своем стремлении принять многочисленные преимущества кремниевых микросхем мы отреклись от ответственности. *Мы позволили пациентам завладеть психбольницей* .

Когда психбольница в руках пациентов, им сложно четко осознать природу собственных проблем. Смотрясь в зеркало, слишком уж просто сконцентрироваться на лучших своих чертах и забыть о недостатках. Когда создатели продуктов, основанных на программном обеспечении, изучают плоды своей ручной работы, они не понимают, насколько эта работа плоха. Они видят только грандиозную мощь и гибкость. Они видят, насколько продукт богат возможностями и функциями. Они игнорируют то, насколько мучительно сложно использовать продукт, сколько часов приходится через силу его изучать или как он унижает и приводит к моральному упадку людей, которым приходится использовать продукт ежедневно.

Мотивы создания этой книги

[4] В компьютерной индустрии термин «разработчик программного обеспечения» употребляется в качестве синонима термина «программист»; то же самое делаю и я в этой книге.

Двадцать пять лет я изобретал и разрабатывал продукты, основанные на программном обеспечении. Многие годы я ломал голову над проблемой сложного в применении программного обеспечения. Наконец, в 1992 году, я прекратил программировать, чтобы посвятить все свое время компаниям-разработчикам, помогая им делать свои продукты более простыми в применении. И случилась удивительная вещь! Я немедленно обнаружил, что, избавившись от потребностей программирования, впервые понял, насколько мощными и всеподчиняющими были эти потребности. Программирование – задача настолько всепоглощающая и сложная, что она доминирует над всеми иными соображениями, включая и заботу о пользователе. Я смог понять это лишь после того, как освободился из капкана программирования.

Совершив такое открытие, я начал понимать, почему программные продукты настолько плохи с точки зрения пользователя. В 1995 году я написал книгу[5] о том, что узнал, и она оказала существенное влияние на разработку некоторых из программ.

Чтобы стать хорошим программистом, необходимо сочувственно относиться к природе и потребностям компьютера. Однако природа и потребности компьютера совершенно чужды природе и потребностям человеческого существа, которому придется в конечном итоге этот компьютер использовать. Создание программного обеспечения требует таких интеллектуальных усилий, так поглощает программистов, что им приходится полностью погружаться в объективно чуждый человеку мыслительный процесс. Для программиста потребности процесса программирования получают приоритет перед любыми потребностями пользователей из внешнего мира, и более того – даже языки этих двух миров конфликтуют.

Процесс программирования ниспровергает процесс создания легких в использовании продуктов по той простой причине, что цели программиста и пользователя коренным образом различаются. Программист желает, чтобы процесс создания протекал гладко и легко. Пользователь желает, чтобы легко и гладко проистекали взаимодействия с программой. Эти две цели практически никогда не приводят к созданию одной и той же программы. В современной компьютерной индустрии программисты отвечают за создание взаимодействий, приятных для пользователя, однако, находясь в безжалостном капкане конфликта интересов, они просто не могут этого делать.

В области программного обеспечения обычно невозможно увидеть результаты, пока работа не завершена, и это значит, что любая рецензия со стороны непрограммиста появится слишком поздно, чтобы дать какой-то эффект. Программное обеспечение для настольных компьютеров имеет дурную репутацию потому, что является исключительно продуктом деятельности программистов; между пользователем и программистами других людей нет. Вещи вроде телефонов и камер всегда имели видимые механические детали, которые делали их легкими для изучения. Однако, как мы установили, если скрестить компьютер практически с чем угодно, стиль поведения компьютера одерживает абсолютную победу.

Ключ к решению этой проблемы – проектирование взаимодействия . Нам нужен новый вид профессиональных проектировщиков взаимодействия, которые станут проектировать поведение программного обеспечения. Сегодня программисты сознательно проектируют «код» программ, но лишь непреднамеренно проектируют взаимодействие с людьми. Они проектируют возможности, но не то, как программа *ведет себя* , *общается* или *уведомляет* . Напротив, проектировщики взаимодействия сосредотачиваются непосредственно на том, как пользователи воспринимают и взаимодействуют с продуктами, основанными на программном обеспечении.

Ремесло проектирования взаимодействия – новое, оно не знакомо программистам, так что – если программисты вообще это признают – ему уделяется внимание лишь после того,

[5] «About Face: The Essentials of User Interface Design» (О лице: основы проектирования пользовательского интерфейса), IDG Books, Foster City CA, 1995, *http:/www.cooper.com/about/face/about_about_face.html.*

как программирование завершено. Но в этот момент уже слишком поздно.

Люди, управляющие созданием продуктов, основанных на программном обеспечении, либо являются заложниками программистов, будучи недостаточно подкованными технически, либо слишком сочувствуют программистам, потому что сами таковыми являются. Пользователи этих продуктов просто не имеют понятия, что эти продукты могут быть приятными в применении и такими же мощными, как любой другой качественно спроектированный инструмент.

Программисты вовсе не злодеи. Они много работают, чтобы сделать свои *программы* легкими в использовании. К сожалению, судят они по себе, так что программы получаются легкими в использовании лишь для других разработчиков программного обеспечения, но не для обычных людей.

Стоимость некачественно спроектированных программ неисчислима. Стоимость времени Джейн и Сунила, стоимость раздражения пассажиров авиалайнера, стоимость жизней пассажиров рейса 965 просто невозможно измерить. А наибольшая растрата – потерянная возможность. Позволяя продуктам приводить нас в отчаяние, увеличивать наши затраты, запутывать, раздражать и убивать нас, мы не пользуемся реальным преимуществом программных продуктов, которые обещали стать наиболее человечными, мощными и приятными творениями из когда-либо выдуманных. Поскольку программное обеспечение сделано из самого податливого материала, оно обладает и потенциалом превзойти ожидания даже самого безумного мечтателя. И требуется лишь разумное сотрудничество проектировщиков взаимодействия и программистов.

Глава 2
Когнитивное сопротивление

Одно дело – осознавать, что проблема существует, и совсем другое – найти ее решение. Важной составляющей процесса поиска решений является язык. За прошедшие годы я разработал ряд полезных терминов и умозрительных моделей. На практике они оказались жизненно важными для формулирования проблемы, которую создают сложные в применении продукты, основанные на программном обеспечении. В этой главе я познакомлю вас с этими терминами и идеями и покажу, как они могут способствовать получению выгоды от применения проектирования взаимодействия в нашем непростом процессе разработки.

Поведение, не связанное с физическими силами

Едва выйдя из индустриальной эры, мы стоим на пороге информационной и держим в руках устаревшие инструменты. В индустриальную эру инженеры справлялись с решением каждой новой проблемы. Работая со сталью и бетоном, они создавали мосты, автомобили, небоскребы и межпланетные ракеты, которые работали хорошо и удовлетворяли своих пользователей. Вступая в информационную эру, мы все больше работаем с программным обеспечением и снова привлекаем лучших инженеров для решения задач. Но теперь все идет не так хорошо. Компьютеры быстры и мощны, программы в целом надежны, однако мы неожиданно столкнулись с фактом существования раздраженных, неудовлетворенных, несчастных и непродуктивных пользователей.

Сегодняшние инженеры не менее талантливы, чем всегда, поэтому я делаю вывод, что они впервые столкнулись с проблемой, которая качественно отличается от всех проблем индустриальной эры. Ведь в противном случае их прежние инструменты работали бы все так же хорошо. За неимением лучшего термина, я назвал эту новую проблему «когнитивным сопротивлением». Это сопротивление, с которым сталкивается человеческий интеллект, пытаясь разобраться в сложной системе правил, изменяющихся динамически.

Взаимодействие с программами имеет высокий показатель когнитивного сопротивления. Взаимодействие с физическими устройствами, пусть даже сложными, как правило, вызывает более низкое сопротивление, потому что механические устройства обычно имеют ограниченное количество состояний в сравнении с количеством внешних воздействий.

Игра на скрипке крайне трудна, однако имеет низкий коэффициент когнитивного сопротивления, поскольку – несмотря на сложность в обращении и изощренность приемов игры – скрипка ни при каких обстоятельствах не входит в «мета»-состояние, в котором звучит, как, скажем, труба или колокольчик. Поведение скрипки всегда предсказуемо, хотя и сложно, и подчиняется физическим законам, несмотря на сложность в обращении. Напротив, микроволновая печь обладает высоким коэффициентом когнитивного сопротивления, потому что десять нумерованных кнопок на панели управления могут существовать в двух контекстах, или режимах. В одном режиме они контролируют интенсивность излучения, а в другом – длительность обработки. Такое серьезное изменение, наряду с отсутствием сенсорной обратной связи, указывающей на изменение состояния печи, и дает в результате высокое когнитивное сопротивление.

Клавиши ЙЦУКЕНГ на печатной машинке не имеют мета-функций. Если нажать на клавишу У, на странице появится буква «У». Если нажать последовательно клавиши СТЕРЕТЬ ВСЕ, на дисплее появятся слова «СТЕРЕТЬ ВСЕ». На компьютере, в зависимости от контекста, могут присутствовать и мета-функции. Будет выполнена операция более высокого уровня, и компьютер *действительно что-то сотрет* . Поведение машины уже не соответствует вашим действиям в отношении один к одному.

Когнитивное сопротивление, подобно трению в физическом мире, не обязательно вредно в небольших количествах, но с нарастанием трения его отрицательные последствия растут экспоненциально. Разумеется, трение представляет собой физическую силу, его можно обнаружить и измерить, тогда как когнитивное сопротивление – инструмент «судебный», и его не следует воспринимать буквально. Однако не забывайте, что такие вещи, как любовь, честолюбие, отвага, страх и правда, будучи вполне реальными, не могут быть обнаружены и измерены. К ним неприменимы и инженерные методы.

Опытные инженеры, создающие микроволновые печи, обычно консультируются со специалистами по человеческому фактору, чтобы спроектировать кнопки, которые легко различать и нажимать. Но специалисты по человеческому фактору всего лишь адаптируют кнопки к глазам и пальцам пользователя, а не к его разуму. Как следствие, микроволновки не обладают значительным физическим «сопротивлением», обладая при этом серьезным сопротивлением когнитивным. Физически легко открыть и закрыть дверцу, нажать на кнопки, однако, учитывая простоту задачи, слишком сложно использовать панель управления для достижения своей цели. Заставить микроволновую печь выполнить задуманное не так просто, хотя наше общее понимание предмета заставляет нас забыть, насколько это сложно в действительности. Сколь многие из нас разогревали что-либо одну секунду или один час вместо одной минуты? Сколь многие готовили что-либо на пятом уровне мощности в течение десяти минут, когда надо было на десятом в течение пяти?

На экране компьютера все переполнено когнитивным сопротивлением. Даже интерфейсы столь простые, как World Wide Web, заставляют пользователей сильнее напрягать мозги, чем любой механизм. Дело в том, что каждая синяя гиперссылка является дорожкой к другому месту в Сети. Вы можете только щелкать по ссылке, а конечная точка маршрута способна меняться независимо от указателя безо всяких предупреждений. Единственное назначение гиперссылки – мета-операции. Когнитивное сопротивление существует именно благодаря гиперсвойству.

Проектирование[6] – слово емкое

[6] В оригинале употребляется слово *design* , обозначающее, в том числе, процесс проектирования программных продуктов. – *Примеч. науч. ред.*

В этой книге говорится о том, что интерактивные продукты должны проектироваться проектировщиками взаимодействия (interaction designers), а не разработчиками программного обеспечения (software engineers). Это утверждение часто вызывает мгновенную неприязнь у программистов, которые всю жизнь занимались проектированием. Более того, эти программисты боятся, что, отнимая у них проектирование, я отнимаю лучший и наиболее творческий аспект их работы, приговаривая их к унылому написанию кода, не способному приносить удовольствие. Это совершенно не так. Их беспокойство происходит из неточной природы термина «проектирование».

Весь процесс создания программного обеспечения пронизан проектированием, начиная с выбора языка программирования и заканчивая выбором цвета грузовика, доставляющего готовый программный продукт. И ни один аспект этого продолжительного и сложного процесса не включает столько проектирования, как собственно программирование. Программисты принимают решения из области проектирования на каждом шаге своей деятельности. Программист должен решить, как одна процедура будет вызывать другие, как будут передаваться, храниться, изменяться данные и информация о состоянии и как обеспечить целостность операций. Все эти решения (и миллионы подобных) – решения проектирования, и успех каждого зависит от способности программиста использовать собственный опыт и здравый смысл.

В этом море проектирования я провожу простую разделительную линию. По одну сторону остаются решения, напрямую влияющие на конечного пользователя. По другую – все прочие решения проектирования. В этой книге, говоря о «проектировании взаимодействия», я говорю лишь о первой категории. Проектирование, не затрагивающее конечного пользователя, я называю «проектированием программы».

Невозможно провести разделительную линию в технических аспектах, и нельзя выразить ее в терминах, знакомых инженерам. Дело в том, что дифференцирующий фактор – антропогенный, а не технический, а правила инженерных наук к людям не применимы. Скажем, проектировщик взаимодействия обычно безразличен к таким вопросам, как выбор языка программирования. Но иногда выбор языка влияет на время реакции системы, наблюдаемое пользователем, что совершенно определенно можно отнести к аспектам взаимодействия, и здесь проектировщику будет что сказать.

Проектирование взаимодействия практически целиком лежит в области выбора поведения, назначения, информации, а также пользовательского представления этих аспектов. Проектирование взаимодействия для конечного продукта – единственная часть процесса проектирования, которую я хочу забрать у программистов и передать в руки людей, специализирующихся на проектировании взаимодействия.

Отношения между программистами и проектировщиками

В мире технологий, где властвуют инженеры, всегда доминировала внутренняя архитектура программы, а проектирование взаимодействия, удобного пользователю, всегда

оказывалось последней задачей, решаемой в свободное время. Одна из целей этой книги состоит в том, чтобы показать выгоды от перестановки приоритетов, и в том, чтобы поставить проектирование взаимодействия во главу угла при создании продуктов, основанных на программном обеспечении.

Большинство программ проектируются случайным образом

Глиняные лачуги и подземные норы проектируются, пусть зачастую неосознанно, в рамках возможностей камня и тростника. Аналогичным образом все программы проектируются в рамках таинственных потребностей языков программирования и баз данных. Самое сильное влияние на проектирование во всех перечисленных материалах оказывает традиция. Разница, и большая, в том, что строитель, проектирующий лачугу, также является и ее основным жильцом, тогда как программисты обычно не используют спроектированные ими приложения.

В большинстве фирм, занимающихся разработкой программного обеспечения, просто нет людей, имеющих представление о проектировании для конечного пользователя. Но есть люди, имеющие более чем серьезное представление о проектировании программ, имеющие свое собственное мнение и личные предпочтения. И потому они делают то, что делают, – проектируют взаимодействие с программой, как для самих себя, выбирая функциональность, которую проще всего и интереснее всего реализовывать, и при этом воображают, что на самом деле проектируют для пользователей. И хотя программисту кажется, что объем выполняемого проектирования очень велик, на деле много всего лишь проектирования *программного* , а проектирование для конечного пользователя почти отсутствует.

Недостаточное проектирование – тоже вид проектирования, поэтому когда кто-либо принимает решения о поведении программы, он принимает на себя роль проектировщика взаимодействия. Когда маркетолог настаивает на включении привлекательной функции в продукт, он проектирует. Когда программист реализует в продукте излюбленный способ поведения, он проектирует.

Разница между проектированием хорошим и непроизвольным – в стиле глиняных домиков – не в применяемых инструментах и приспособлениях, но в мотивации. Настоящий проектировщик взаимодействия принимает решения, исходя из задач пользователя. Эрзац-проектировщики принимают решения, исходя из произвольного числа случайных соображений. Личные предпочтения, знакомство с предметом, страх перед неизвестностью, указания от Microsoft, ошибочные замечания коллег – все эти факторы играют на удивление серьезную роль. Впрочем, чаще всего решения эрзац-проектировщиков склоняются в сторону пути наименьшего сопротивления.

Проектирование «взаимодействия» против проектирования «интерфейса»

Я предпочитаю термин «проектирование взаимодействия» термину «проектирование интерфейса», поскольку слово «интерфейс» предполагает, что код находится в одном месте, люди в другом, а интерфейс между ними позволяет обмениваться сообщениями. Подразумевается, что именно интерфейс несет ответственность за потребности пользователей. Последствия изоляции проектирования на уровне интерфейса таковы – программисты начинают делать выводы примерно такого характера: «Я могу писать, как мне угодно, потому что „интерфейс“ появится уже после того, как я закончу». Проектирование откладывается до завершения программирования, когда уже слишком поздно.

Дизайн интерфейса позволяет придать определенный вид уже существующему поведению системы, так и гунна Аттилу можно одеть в костюм от Армани. Например, в генераторе отчетов дизайн интерфейса устранит ненужные границы и прочие визуальные помехи из таблицы с цифрами, выделит цветом важные поля, обеспечит качественный отклик на действия пользователя и т.д. Это лучше, чем ничего, но далеко не достаточно.

Microsoft вкладывает многие миллионы долларов в проектирование интерфейсов, но продукты этой компании так и не снискали любви пользователей.

«Поведенческое проектирование» указывает, как элементы программы должны действовать и передавать информацию. Продолжая пример, можно сказать, что проектирование поведения указывает, какие инструменты можно применять к таблице отчета, как включать в отчет средние или итоговые показатели. Проектировщики взаимодействия также работают от общего к частному, начиная с целей, которых пытается достичь пользователь, но, не забывая о более широких потребностях бизнеса, ограничениях технологии и подчиненных задачах.

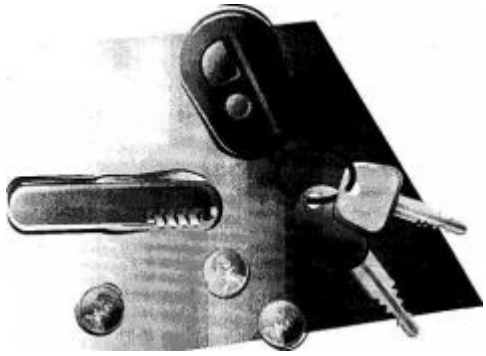
Можно углубиться еще далее, в область, которую мы называем «концептуальным проектированием». Концептуальное проектирование рассматривает вещи с точки зрения их возможной ценности для пользователей. В нашем примере концептуальное, проектирование может выявить, что изучение таблицы с данными – второстепенная задача, а реальная цель пользователя – в том, чтобы отследить тенденции. Отсюда следует, что надо создавать вовсе не генератор отчетов, а анализатор тенденций. Чтобы обеспечить пользователей ощущением могущества и удовлетворения, необходимо сначала думать *концептуально* , затем в терминах *поведения* и лишь в последнюю очередь – в терминах *интерфейса* .

Отличительные черты продуктов, основанных на программном обеспечении

Когнитивное сопротивление присуще всем продуктам, основанным на программном обеспечении, независимо от их примитивности, и делает эти продукты гораздо более сложными в использовании, чем аналогичные продукты механической эры. Вот, для примера, содержимое моих карманов: несколько монет, швейцарский армейский нож, ключи от автомобиля. Нож – в чистом виде продукт индустриальной эры: вы видите, как он сделан, как работает и как его можно использовать, – достаточно бросить на него поверхностный взгляд и покрутить в руках. Раскрыв лезвие ножа, вы можете увидеть, что оно острое, и представить, какими режущими свойствами обладает.

У ножа целых шесть лезвий, зубочистка и пинцет. Назначение всего становится очевидным сразу. Я могу с легкостью понять, как управляться с ножом, благодаря тому, как он подходит к моей руке и пальцам. Одно удовольствие использовать этот нож.

Не требующая ключей система доступа на брелоке от автомобиля – совершенно иная зверушка. Здесь всего две кнопки, так что с точки зрения манипуляций она проще ножа. Как только черный и гладкий пластиковый брелок оказывается в моей руке, пальцы естественным интуитивным образом обнаруживают две кнопки, назначение которых очевидно – нажми, чтобы активировать. Да, но за кнопками кремний, а не сталь, так что управляться с ними сложнее, чем может показаться.



Крупная кнопка запирает автомобиль и одновременно включает сигнализацию. Второе нажатие отключает сигнализацию и отпирает двери автомобиля. Вторая кнопка, поменьше, обозначена словом «Panic». Если нажать на нее, автомобиль в течение нескольких секунд издает тихие трели. Если удерживать кнопку нажатой дольше, на смену тихим трелям

приходит вой автомобильной сигнализации, во все свои сто децибел грохочущей, чирикающей, орущей на целый километр окрест, что какой-то болван, то есть я, только что сделал нечто ужасно глупое. Что еще хуже, после включения сигнализации маленькое пластиковое устройство становится неактивным, и дальнейшие нажатия на кнопки уже не дают никаких результатов. Единственный способ остановить громогласное распространение информации о моей очевидной глупости – подойти к моей ужасающе громкой машине, под уничижающими взглядами прохожих открыть водительскую дверь ключом, затем вставить ключ в зажигание и повернуть его. Кроме шуток, я чувствую себя идиотом. Если бы мою машину просто вскрыли и ограбили, я бы огорчился и почувствовал себя оскорбленным, но во всяком случае не дураком.

В предыдущей своей книге я утверждал, что первоочередная цель всех пользователей компьютеров заключается в том, чтобы не чувствовать себя дураками. Далее, я предположил, что хорошие интерфейсы позволяют избегать ситуаций, когда ручка катапультирования может оказаться среди других часто используемых органов управления. Это классический пример устройства, которое заставляет пользователей почувствовать себя глупо, так как рычаг катапульты находится прямо у них перед глазами. Случайно потянув за этот рычаг, человек попадает в сложное положение, эквивалентное появлению на работе без штанов. Мой швейцарский армейский нож просто не позволяет проделать что-то подобное.

Мне *сложно придумать* причину, по которой кто-либо захотел бы воспользоваться любой из функций второй кнопки, и к тому же совершенно непонятно, почему создатели пульта не воспользовались замечательной возможностью предоставить мне *действительно* нужные и полезные функции.[7]

Мне очень приятно, что в комплект поставки моего автомобиля входит сигнализация, однако есть много ситуаций, когда я хотел бы просто запирать машину, не ставя на сигнализацию. Когда я заглядываю в местную кофейню, чтобы выпить кофе, мне не нужен тот же уровень защиты, как, скажем, в аэропорту. Мне было бы очень удобно иметь возможность запирать и открывать автомобиль с дистанционного пульта, не включая сигнализацию. Это было бы весьма полезно в поездках по местным магазинам или когда я высаживаю детей у школы.

Не менее полезной и приятной была бы еще и поддержка более безопасной запирающей системы. Время от времени, возвращаясь к закрытому автомобилю, я обнаруживаю, что в мое отсутствие он перестал таковым быть. Это происходит, когда человек на похожем автомобиле того же производителя паркуется рядом со мной на большой стоянке. Когда этот человек нажимает на кнопку, запирая свой автомобиль, он также дает сигнал на отпирание моего, другими словами, отключает сигнализацию и открывает мою машину проходящим мимо антиобщественным личностям. Такой сценарий опаснее всего именно в тех ситуациях, когда он наиболее вероятен: на крупных городских парковках, в аэропортах, где моя машина может простоять несколько часов или даже дней, подверженная случайному срабатыванию системы дистанционного запирания. Несомненно, если бы я мог запереть и поставить машину на сигнализацию таким образом, чтобы только мое собственноручное вмешательство и вставленный в дверь металлический ключ могли разблокировать ее, это стало бы достойным применением технологии. Конечно, я знаю, что такой способ существует, ведь именно так выключается сработавшая сигнализация. К сожалению, проектировщики системы постарались, чтобы большая кнопка на чьем угодно

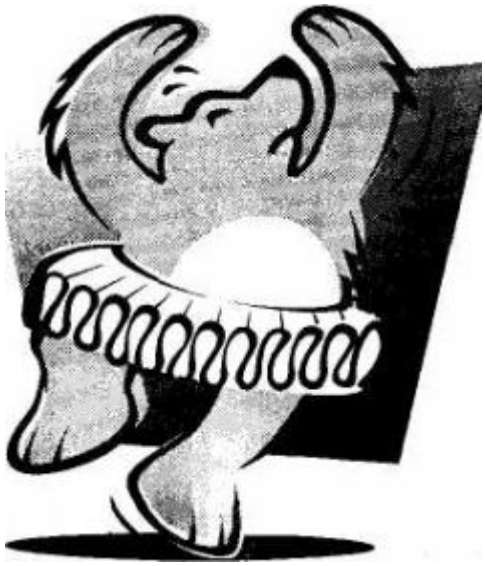
[7] Мне много раз говорили, что среди женщин эта функция пользуется спросом – как сдерживающее средство, помогающее противостоять преступникам на темных парковках, однако всякий раз это говорил технически подготовленный мужчина, который сам никогда не воспользовался бы этой кнопкой. К своему большому удивлению, недавно я прочел в Wall Street Journal о настоящем применении кнопки паники. В Йосемитском национальном парке дикий медведь пристал к одной семье на кемпинге. Он принялся тормошить автомобиль, пытаясь добраться до запертой внутри еды. Мать семейства нажала на кнопку паники, и сирена в конечном итоге отпугнула медведя. Возможно, имеет смысл назвать эту маленькую кнопку «Репеллент от медведей».

брелоке разблокировала мой автомобиль, как бы я ни запирал его.

Швейцарский армейский нож сложен и насыщен возможностями, причем некоторые из них весьма хитроумно спрятаны, однако изучение и применение ножа – процесс простой, предсказуемый, интуитивный. Дистанционное управление замками сложно, связано с проблемами, способно мгновенно поставить меня в неприятную ситуацию. Устройство не делает того, что мне нужно, и не позволяет мне добиться нормального и приемлемого для меня уровня контроля над собственным автомобилем. Короче говоря, взаимодействие с этой системой вызывает отвращение. Она попросту *плоха* , и я ее ненавижу.

Танцующий медведь

С другой стороны, если мне придется сделать выбор между ножом и дистанционным управлением замками, я выброшу нож без раздумий. Впервые испробовав дистанционное управление замками, я уже не мог представить себе автомобиль *без* него. Это самая удобная функция в моем автомобиле, я использую ее чаще любой другой – в десять раз чаще, чем нож. Несмотря на свой неубедительный и неуклюжий дизайн, это все же замечательная вещь. Можно провести аналогию со здоровенным медведем, которого человек выводит на цепи на городскую площадь и за скромные пожертвования заставляет танцевать. Горожане собираются поглазеть на диковинку – на то, как массивный, громоздкий зверь неуклюже переступает с лапы на лапу. Танцует медведь просто ужасно, но *чудо не в том, что он танцует хорошо, а в том, что вообще танцует* .



Чудо не в том, что дистанционное управление замками работает хорошо, а в том, что оно вообще работает. Я готов смириться с проблемами взаимодействия, чтобы воспользоваться преимуществами удаленного доступа к своему транспорту.

Удивительные дары кремния столь неодолимо притягательны, что мы готовы легко примириться с сопутствующими затратами. Попав на необитаемый остров, вы не станете возражать, если пришедший на помощь корабль окажется ржавым остовом, изъеденным течами и кишащим крысами. Разница между наличием программного решения проблемы и отсутствием решения вообще настолько велика, что мы принимаем любые испытания и трудности, сопутствующие этому решению.

Неподатливость проблемы происходит не из сложности создания более совершенных взаимодействий. Она происходит из нашей всеобщей готовности принимать некачественные взаимодействия, как неизбежное. Завидев проржавленный корабль, мы не интересуемся, какие на нем удобства, а просто прыгаем на борт и счастливы тем, что получили.

Специалистам в области программного обеспечения только и остается, что чувствовать

себя комфортно, сталкиваясь с продуктами, имеющими высокое когнитивное сопротивление. Они гордятся своей способностью работать, несмотря на превратности судьбы. Обыкновенные люди, являющиеся новичками в использовании этих продуктов, не имеют специальных знаний, позволяющих судить, можно ли избавиться от когнитивного сопротивления. Вместо этого они полагаются на подсказки компьютерных «ботаников», которые пожимают плечами и говорят, что «надо быть компьютерно грамотным», чтобы использовать продукты, основанные на программном обеспечении. Программисты валят всю вину на технологии, объясняя пользователям, что сложность взаимодействия – неотъемлемое свойство компьютеров, и она неизбежна.

Это неправда. Сложных взаимодействий вполне можно избежать.

Когнитивное сопротивление порождается не технологиями, а людьми, которые ими владеют. Они хозяева, потому что умеют думать на языке кремниевых микросхем и воображают, будто все думают так же. Они создают технологические артефакты, взаимодействие с которыми осуществляется на том же языке, который применялся в разработке. Они лучше сделают автомобиль из раскаленного металла и скрежещущих шестерен, чем отделают его кожей и деревом. Они – инженеры и больше думают о шестернях, чем о коже, поэтому интерфейс с человеком-пользователем выражается в терминах «реализации» продукта, и продукты, проектируемые подобным образом, я называю «моделями реализации».

Стоимость дополнительных возможностей программного обеспечения

Большинство фирм-разработчиков программного обеспечения не представляет, как сделать программы простыми в применении, но, несомненно, знает, как начинять их возможностями, а потому именно этим и занимается.

В физических объектах, каковым является мой швейцарский армейский нож, существует естественное противодействие умножению дополнительных возможностей. Производитель вынужден увеличивать затраты, добавляя новое лезвие или приспособление к ножу. Создателю ножа это известно, поэтому каждая из предложенных возможностей подвергается жесткому анализу, прежде чем попасть в конечный продукт. В терминах инженерных наук это явление называется отрицательной обратной связью – здесь внутренние силы стремятся к стабильности и равновесию. К примеру, трение покрышек вашего автомобиля создает отрицательную обратную связь в рулевой системе, поэтому если отпустить руль, он выравнивается в изначальное положение.

В экономике продукции, основанной на программном обеспечении, доминирует иная система. Поскольку разнообразные функции добавляются не в осязаемую сталь, медь или пластик, а в неосязаемый программный код, руководителям старой закалки кажется, что дополнительные возможности почти бесплатны. Им кажется, что функциональную мощь программ легко наращивать и «улучшать».

Прямо сейчас я слушаю диск Джимми Бафета при помощи оптического привода своего компьютера. Небольшая программа, проигрывающая диск, предлагает мне множество функций: я могу перейти к предыдущей композиции, к следующей, к любой выбранной, могу задать особый порядок воспроизведения, запланировать проигрывание в течение определенного времени, зациклить дорожку, просмотреть информацию о Бафете в сети, добавить альбом в свою коллекцию, создавать примечания к различным композициям, получать названия песен из базы данных в Интернете, просматривать информацию о диске, создавать список любимых композиций с диска и много чего еще. Все это очень приятные возможности, и я, наверное, не стал бы от них избавляться, однако в целом они крайне затрудняют понимание и использование программы. Кроме того, если звонит телефон и мне надо быстро приостановить воспроизведение диска, я не могу найти нужную функцию, потому что она похоронена среди тех самых многочисленных – бесплатных – функций. Для меня эти функции не «бесплатны». Какой-то горе-инженер решил, что окажет мне

любезность, если добавит все эти бесплатные функции, но я бы предпочел простое средство воспроизведения с легкодоступной кнопкой паузы.

Что касается системы дистанционного управления замками, я серьезно сомневаюсь, что хоть один из ее разработчиков задался вопросом, какие функции нужны и сколько их должно быть. Уверен, что вместо этого какой-то из младших сотрудников выбрал стандартную микросхему, случайно имевшую два канала. Воспользовавшись одним каналом для реализации функций запирания и отпирания, он обнаружил, что есть еще один бесплатный канал. Этот инженер, возможно, находясь под влиянием очень активного, но некомпетентного менеджера по маркетингу, на ходу выдумал аргументы в пользу того, что ручное включение сигнализации может послужить какой-то цели. Он был горд, что смог обеспечить дополнительную функциональность без очевидных затрат.

Полноценный микропроцессор в ключе вашего автомобиля, в микроволновой печи или мобильном телефоне обходится дешевле, чем отдельные микросхемы и электронные компоненты. Так новая экономика влияет на проектирование продукта. Отрицательная обратная связь стоимости производства сдерживает добавление новых физических органов управления, но не сдерживает процесс добавления функций и возможностей программного обеспечения. Разработчикам программ кажется, что новые функции практически бесплатны, поэтому любая предложенная функция *считается* хорошим вложением, если не доказано обратное. При отсутствии сдерживающих факторов продукт быстро раздувается от ненужных функций, что усложняет и запутывает его с точки зрения пользователя. Все эти функции, конечно же, выставляются как абсолютно необходимые преимущества, а кроме того, по-прежнему сохраняется основная и действительно востребованная функция. Вот что такое танцующий медведь.

В случае с настольными компьютерами отсутствие обратной связи не менее опасно. Создатель программы воображает, будто можно добавлять любые функции по собственному желанию, и они будут «бесплатны», если доступны посредством стандартной клавиатуры и мыши. Они наполняют экраны сотнями загадочных пиктограмм, кнопок, элементов меню, и каждая функция в конечном итоге требует работы с клавиатурой или щелчка кнопкой мыши. Как же пользователю отличить мелкие, незначительные функции от функций, приводящих к серьезным отрицательным последствиям?

Практически каждый коммерческий программный продукт усложнялся с каждой новой версией. В ходе эволюции продукта добавляются функции и возможности, а потому в интерфейсе появляются новые органы управления. На жаргоне программистов это явление называется «bloatware» – распухшее программное обеспечение. Продукты вроде Lotus Notes, Adobe Photoshop, Intuit Quicken и Microsoft Word настолько плотно напичканы возможностями, что пользователей сбивает с толку беспорядочное нагромождение функций, редкие из которых используются эффективно, если вообще используются. Между тем, мириады малополезных возможностей вытесняют немногие действительно необходимые функции.

В корпоративных программах проблема проявляется еще более остро, чем в приложениях для конечных пользователей. Такие фирмы, как Oracle, PeopleSoft, ADP, SAP, Siebel, создают сложное программное обеспечение, необходимое для корпоративной деятельности. Эти продукты крайне сложны, невразумительны и перегружены возможностями. Каждое ежегодное обновление добавляет множество новых функций, но по-прежнему оставляет уже имеющиеся функции непонятными и неподвластными для людей, не прошедших месяцы суровых тренировок.

Апологеты и уцелевшие

Танцующие медведи уже везде. Невероятная мощь компьютеров означает, что очень немногие могут себе позволить игнорировать их. Даже не имея настольного компьютера, вы, вероятно, владеете видеомагнитофоном и кредитной картой, а это продукты, основанные на

программном обеспечении. Невозможно просто сказать «не буду пользоваться компьютерами». Они не просто дешевеют, а дешевеют со страшной скоростью, стремясь к повсеместному распространению, и приобретают свойства одноразовых вещей. Многие распространенные продукты, которые мы считаем механическими (или электронными) уже не производятся без процессоров. Автомобили, стиральные машины, телевизоры, пылесосы, термостаты, лифты – вот лишь несколько хороших примеров.

В индустриальную эру польза от устройства была пропорциональна сложности работы с ним, однако в век информации подобная связь отсутствует, а сложность управления возрастает быстрее, чем польза. Старомодный механический будильник всегда считался простым устройством. Совладать с современным будильником, основанным на программном обеспечении, может быть, сложнее, чем с автомобилем.

Высокое когнитивное сопротивление раскалывает людей на два лагеря. Оно заставляет их впадать в отчаяние и чувствовать себя глупо из-за неудач либо чувствовать эйфорию от способности преодолевать крайне серьезные препятствия. Столь мощные эмоции делают из людей «уцелевших» и «апологетов». Они либо воспринимают когнитивное сопротивление как стиль жизни, либо уходят в подполье, принимая его как неизбежное зло. Этот раскол становится все острее.

* * *

Вторую группу я называю апологетами потому, что ее участники прилагают максимум усилий, чтобы оправдать свое преклонение перед танцующим медведем. Подобно фанатам политических партий, в глупых шляпах и с бестолковыми плакатами, они превозносят преимущества, в то же время с беззастенчивым фанатизмом преуменьшая недостатки. Практически все программисты попадают в эту категорию, и их собственные интересы делают такую мотивацию понятной, однако просто удивительно, сколь многие несведущие в технике пользователи оправдывают своих угнетателей, находясь под ежедневным воздействием некачественных продуктов: «Это же легко. Достаточно запомнить, что надо нажать эти две клавиши, затем назначить системе корректное имя. Если я забуду имя, система позволяет мне найти его». Они не понимают, насколько нелепо, что система «позволяет найти». Почему бы компьютеру и не заниматься поиском или запоминанием? Апологеты – это люди, защищающие компьютер за то, что он позволяет выполнять задачи, которые раньше выполнять было невероятно сложно. Они показывают на медведя и восклицают: «Смотрите, он же танцует!»

Апологеты напоминают мне жертв стокгольмского синдрома. Речь о заложниках, которые влюбляются в своих захватчиков, безо всякой иронии или же признаков

рационального мышления заявляя: «Он замечательный человек. И даже разрешает нам пользоваться туалетом».

«Продвинутый пользователь» – лишь кодовое название апологета. Независимо от сложности взаимодействия с системой, независимо от того, насколько невразумительна ее функциональность, апологет будет безошибочно указывать на мощность и возможности технического средства, блаженно игнорируя сложность его реального применения.

Одна из моих коллег, занимающаяся мобильными телефонами, пожаловалась, что инженеры затрудняют использование мобильных телефонов, встраивая в них массу редко востребуемых возможностей. Она сказала, что мобильные телефоны – это «мокрые собаки». Когда я спросил ее, как следует понимать эту метафору, она пояснила: «Нужно очень и очень сильно любить мокрую собаку, чтобы повсюду таскать ее за собой».

Просто удивительно, как компьютеры привлекают очень умных, инициативных людей в невероятных количествах. Похоже, этих же самых людей привлекает опасный и сложный спорт – скайдайвинг, пилотирование, подводное плавание, игра на бирже, скалолазание. Все эти занятия требуют скрупулезной подготовки, и малейшая невнимательность может закончиться катастрофой. Однако не имей эти увлечения огромного очарования, не будь они неотразимо притягательны, их приверженцы могли бы с равным успехом смотреть телевизор, ведь правда? Притягательность как раз в том, что все эти занятия сложны. Она в сложности решения интеллектуальной задачи, не допускающей ошибок. Легко представить, как вспотевший, утомленный альпинист потягивает Gatorade и, улыбаясь, произносит: «Последний участок был совершенно отвесный, пару раз я чуть не сорвался. Мышцы просто звенели от напряжения». Ему *нравится* , когда это тяжело! Чем тяжелее, тем лучше! Потому-то он и занимается этим!

Компьютеры вдохновляют людей тем же образом, потому что предлагают такие же крутые, беспощадные испытания. И если вы не в идеальной форме, компьютер оставит вас хныкать в придорожной пыли. Легко представить утомленного программиста, который потягивает колу, ухмыляется и говорит: «Да, подпрограмма поиска приводила к сбою, но только когда размер кучи превышал 64 килобайта, потому что в противном случае кэш не использовался. С большим трудом обнаружил!» Он получает *удовольствие* !

Так и появляются апологеты. Они наслаждаются суровыми испытаниями, их неумолимой природой. Им нравится работать в среде, где особые способности делают их не такими, как все другие люди. Альпинист защищает крутизну и сложность подъема. Компьютерный энтузиаст защищает непрозрачность и затрудненность взаимодействия с программным обеспечением.

* * *

На другом полюсе находятся уцелевшие. Они нередко понимают: что-то в корне неправильно, но не могут сказать, что именно. Они плохо понимают в компьютерах и взаимодействиях, но понимают, что существует проблема. Они знают, что такое *сложно* и что такое *легко* , и очень хорошо понимают, что *компьютеры – это сложно* .

Однако, как и все остальные, они не могут просто отказаться от компьютеров; компьютеры нужны для работы. Эти люди скрежещут зубами и поневоле мирятся с манерами программ, похожих на танцующих медведей. Они не знают, что компьютеры могут вести себя лучше, но знают, что каждый сеанс работы с компьютером заставляет их чувствовать себя менее уверенно, чем обычно. Подобно феодальным крестьянам средних веков, они не в силах изменить свой статус или хотя бы увидеть глубину собственных лишений, однако четко осознают, что их угнетают.

Апологеты говорят: «Смотрите, что я могу сделать при помощи компьютера!» Уцелевшие говорят: «Видимо, я слишком глуп, чтобы понять эти новомодные машины». Апологеты говорят: «Взгляните-ка! Танцующий медведь!» Уцелевшие говорят: «Мне нужно нечто танцующее, и медведь, судя по всему, – лучший из имеющихся вариантов». Уцелевшие – это подавляющее большинство людей, которых не впечатляет обретенная мощь, но которых весьма впечатляет, насколько глупо они себя чувствуют, общаясь с компьютерами.

Разумеется, практически каждый в компьютерной индустрии и родственных отраслях, создающих компьютерные продукты и услуги, однозначно происходит из лагеря апологетов. Их поведение отражает их точку зрения. Они всегда защищают свои продукты, подчеркивая их мощь и возможности. А вот сталкиваясь с вопросами из человеческой области, они – подобно политикам – склонны красочно расписывать новые функции и возможности продукта и указывать на количество пользователей, вместо того чтобы отвечать на предложенные вопросы. Они игнорируют неуклюжесть танца, превознося сам его факт.

Невероятная скорость роста Интернета и легкость доступа к нему привели к вторжению новоявленных апологетов и уцелевших в мир компьютеров. Апологеты с энтузиазмом указывают на многочисленные службы и источники информации, доступные в Интернете. Тем временем уцелевшие гипнотизируют экраны своих компьютеров, теряясь в догадках, как найти что-нибудь полезное. Они могут до бесконечности ждать загрузки ненужных изображений на сайтах, а потом все равно теряться в сложных иерархиях ненужной информации. Всемирная паутина – вероятно, самый большой из известных человеку танцующих медведей.

Наша реакция на когнитивное сопротивление

На когнитивное сопротивление люди, включая и поборников, в большинстве своем реагируют одинаково. Они берут необходимый минимум, а все остальное игнорируют. Каждый пользователь осваивает наименьший из возможных набор функций, позволяющий решать рабочие задачи, а от остальных функций отказывается. Апологет может с гордостью рассказывать, что его наручные часы способны синхронизироваться с календарем настольного компьютера, однако почему-то забывает упомянуть, что в последний раз

пользовался этой функцией полгода назад. Если прижать его по этому поводу, он займет оборонительную позицию – на то он и апологет.

В моем новом домашнем развлекательном комплексе буквально тысячи функций. Я не апологет, но определенно испытываю пристрастие ко всяким техническим штучкам. Я научился использовать некоторые из даровых возможностей комплекса, но их слишком сложно использовать эффективно. К примеру, телевизор имеет функцию картинки в картинке (КВК). В правом нижнем углу экрана появляется дополнительный маленький экран, в котором идет изображение с другого канала. Это полностью программное решение, и оно управляется кнопками дистанционного пульта. В теории функция может оказаться удобной, если я хочу следить за футбольным матчем, но смотреть при этом фильм на основном экране. Когда продавец демонстрировал эту функцию, она показалась мне весьма полезной. Загвоздка в том, что эту функцию слишком сложно использовать. Когнитивное сопротивление таково, что я не могу в достаточной степени овладеть ею, чтобы оправдать ее присутствие. Гораздо приятнее смотреть один канал – как в старые времена, когда других вариантов и не было из-за ограничений технологии. Другим членам моей семьи не пришло в голову использовать функцию картинки в картинке хотя бы однажды; они делают это разве что случайно. Иногда я прихожу домой и вижу, что кто-то смотрит телевизор с картинкой в картинке. Как только я вхожу в комнату, меня сразу просят отключить КВК.

Мой телевизор обладает экраном с диагональю в 55 дюймов, звуковой системой Dolby, а сигнал принимает со спутника, но я и члены моей семьи *используем* его точно таким же образом, как старый ящик Motorola с диагональю 19 дюймов в 1975 году. Все эти новые возможности остаются невостребованными.

Можно предсказать, какие возможности любой новой технологии будут задействованы, а какие нет. Востребованность функции обратно пропорциональна количеству действий, необходимых для управления этой функцией. Иначе говоря, больший по размеру, более яркий экран моего нового телевизора не требует с моей стороны никаких телодвижений, а потому используется непрерывно, если телевизор включен. Экраном я вполне доволен. Спутниковая система – весьма необходимый танцующий медведь, поэтому я мирюсь со сложностью переключения источника сигнала, чтобы примерно раз в неделю смотреть спутниковые передачи. Никто другой в моей семье не мог разобраться, как смотреть спутниковые каналы, пока я не приклеил ламинированную шпаргалку на кофейный столик: она содержит перечень переключателей, кнопок и режимов. Функция КВК управляется сложной системой из более чем десятка клавиш, мало того – взаимодействие с ней не очевидно, а ее поведение неприятно. После первых же попыток я забросил эту функцию, как и все остальные.

Такого рода когнитивное сопротивление, приводящее к фильтрации возможностей, можно обнаружить в любом офисе или доме, где используются продукты, основанные на программном обеспечении.

Демократизация власти потребителя

Исторически так сложилось, что более сложные механические устройства требовали более серьезной подготовки операторов. Крупные машины всегда надежно охранялись, и доступ к ним имели только подготовленные профессионалы в белых лабораторных халатах. Информационная эра все изменила, и теперь мы ожидаем, что любители смогут управиться с технологией гораздо более сложной, чем все технологии наших предков.

Наши инструменты и системы все чаще содержат мозги из кремния, все чаще попадают в руки неподготовленных новичков. Двадцать пять лет назад междугородние звонки обрабатывал специальный оператор, и делал он это по нашей устной просьбе. Сегодня самые сложные международные звонки совершает сам звонящий, неподготовленный дилетант, нажимающий кнопки.

Пару десятилетий назад даже бензоколонки находились в ведении специально

подготовленного персонала. Сегодня же каждый человек должен уметь обращаться с бензоколонкой. Сюда входит и способность самостоятельно произвести финансовую операцию по кредитной или дебетовой карте. Двадцать лет назад взаимодействие с банком осуществлялось только через подготовленного кассира. Сегодня вы сами работаете со своим банком на местной заправке или посредством банкомата.

Инженерный процесс не отличает создание сложной системы, предназначенной для подготовленных профессионалов, управляющих этой системой за деньги, от создания системы, с которой будет иметь дело дилетант. В инженерном процессе нет способов учесть человеческий фактор, он сосредоточен на вопросах реализации. Из чего это сделано? Как будет происходить сборка? Какие компоненты интерфейса нужны, чтобы ввести все возможные переменные?

Виноват пользователь

Программное обеспечение используется преимущественно в деловом контексте, так что многие жертвы некачественных взаимодействий получают деньги за свои страдания. Они вынуждены использовать программы для работы и поэтому не могут *отказаться* от программ, а могут лишь терпеть, пока хватает сил. Им приходится подавлять раздражение и не обращать внимания на ощущение собственной глупости, причина которого как раз в программном обеспечении.

Многие годы я наблюдаю, как десятки менеджеров индустрии программного обеспечения рисуют по сути дела одну и ту же диаграмму, представляющую их взгляд на рынок высоких технологий. На диаграмме представлена пирамида (некоторые рисуют ее вверх ногами), разделенная на три слоя. Каждый слой помечен невинно звучащим словосочетанием. К этой пирамиде каждый руководитель пририсовывает облако с обозначением сегмента рынка, который компания намеревается завоевать. При этом название каждого из слоев – эвфемизм, завуалированный оскорбительный намек, словно кодовая фраза ханжи, посредством которой он исключает людей из числа достойных посещать клуб для избранных. Вот эти три эвфемизма для пользователя: «неподготовленный», «компьютерно образованный» и «продвинутый».

«Неподготовленный пользователь» – принятое в этой индустрии обозначение «пользователя глупого и некомпетентного». Конечно, этих людей заставляют почувствовать себя глупыми, но они таковыми не являются. Виновато же некачественное проектирование взаимодействия. Апологеты компьютерной индустрии не принимают неподготовленных пользователей в расчет, но это противоречит здравому смыслу. С какой это стати продавцу программ отказываться от львиной доли рынка? А с такой, что это снимает вину за провал с руководителей и разработчиков и переносит ее прямо на плечи невиновных пользователей.

Словосочетание «компьютерно образованный пользователь» в действительности обозначает человека, которому так много раз делали больно, что толщина рубцовой ткани

уже просто не позволяет ему чувствовать боль. Образование в области компьютеров означает, что если программа потеряла документ, вы не паникуете, как Джейн из главы 1 «Загадки века информации», но начинаете медленный, самостоятельный, совершенно ненужный поиск файла в иерархической файловой системе, и при этом не жалуетесь. Одно из свойств компьютерного образования – оно подобно анестезии, медленно и плавно погружающей пациента в бессознательное состояние. Нет особого смысла постоянно ныть и жаловаться на программу, которая составляет обязательную и постоянную часть вашей работы. Большинство людей даже не осознает своих титанических усилий, направленных на компенсацию недостатков инструментов, основанных на программном обеспечении.

Апологеты в основном считают компьютерную образованность чем-то вроде символа определенных достижений, как награда за отличную стрельбу. В действительности же более уместна аналогия с Пурпурным Сердцем, официальным признанием ранения, полученного в бою.

Продвинутые пользователи – это просто апологеты. Техноэнтузиасты, сумевшие подавить лучшие свои инстинкты, чтобы стать полезными потребителями продукции с высоким когнитивным сопротивлением. Они гордятся испытаниями, словно штурмуют скалу в Йосемитском заповеднике.

Программный апартеид

В Голливуде шутят, что можно обратиться к незнакомцу в бакалейной лавке и спросить, как продвигается его сценарий. И незнакомец без промедления ответит: «Отлично! Я только что реструктурировал второе действие, чтобы усилить напряжение событий!» Та же шутка теперь верна и в отношении Кремниевой Долины. Вы можете пристать к незнакомке в очереди в кофейне и поинтересоваться, как продвигается работа над веб-сайтом. И она ответит, глазом не моргнув: «Отлично! Я только что реструктурировала фреймы, чтобы улучшить навигацию!»

Здесь, в Долине, мы забываем, насколько мы отличаемся от остального населения, потому должны часто напоминать себе об этом. Здесь средний пользователь программного обеспечения на деле далеко не средний.

Программисты обычно работают в окружении технически равноценных коллег, в анклавах вроде Кремниевой Долины, трассы 128 в пригороде Бостона, Научного Треугольника в Северной Каролине, Редмонда в Вашингтоне и Остина в Техасе. Разработчики программного обеспечения постоянно встречаются с коллегами в магазинах, в ресторанах, когда отвозят детей в школу, на отдыхе, в то время как их контакт с

раздраженными пользователями компьютеров ограничен. Более того, редкие всплески досады пользователей, не направленные ни на кого конкретно, компенсируются частыми проявлениями энтузиазма со стороны продвинутой элиты. Мы забываем, насколько далеки от неудовольствия, вызываемого у других жителей страны (не говоря уже обо всем мире) работой с интерактивными инструментами.

Мы бросаемся словами «компьютерное образование», подразумевая, что человеку требуется достичь определенного уровня подготовки, чтобы пользоваться компьютерами. Мы считаем это простым требованием, разумным и правильным. Мы полагаем, что не так уж глупо требовать от пользователей приобретения начальных познаний в устройстве машин, если благодаря этому первые смогут наслаждаться преимуществами последних. И все же это *слишком* серьезное требование. Наличие базы компьютерно образованных клиентов сильно облегчает процесс разработки, в этом нет сомнений, но затрудняет рост и движение к успеху для индустрии и общества. Апологеты возражают, что для вождения автомобиля требуется подготовка и сдача экзамена на знание правил дорожного движения, однако они упускают из виду, что ошибка при вождении автомобиля часто приводит к гибели людей, тогда как ошибка при работе с программой обычно имеет менее суровые последствия. Не будь машины столь смертоносными, люди учились бы водить так же, как осваивают Excel.

Здесь присутствует и другой, более коварный эффект. Подобный подход проводит демаркационную линию между имущими и неимущими. Если владение компьютером необходимо, чтобы преуспеть на американском рынке труда и стать кем-то, помимо менеджера предприятия быстрого питания, то сложность изучения интерактивных систем вынуждает многих людей браться за рабский труд, не позволяя добиться более производительных, уважаемых, более прибыльных позиций.

Пользователь не должен получать компьютерное образование, чтобы решать посредством компьютера простые каждодневные задачи начального уровня. Пользователь не обязан обладать цифровой сноровкой, чтобы получать электронную почту или совладать с видеомагнитофоном и микроволновой печью. Более того, пользователи не должны получать компьютерное образование, чтобы применять компьютеры в корпоративных приложениях, где они уже владеют предметной областью. Так, бухгалтер, сведущий в общих принципах бухучета, не должен получать компьютерное образование, чтобы применять компьютер в своей работе. Ему должно хватать познаний в предметной области.

По мере того, как экономика все крепче базируется на информации, мы, сами того не желая, создаем в обществе раскол. Высший класс состоит из тех, кто знает различие между такими понятиями, как «оперативная память» и «жесткий диск». Низший класс – из тех, кто считает такое различие надуманным. Ирония в том, что различие и впрямь *не принципиально* для всех, кроме совсем закоренелых инженеров. И все же большинство современных программ вынуждают пользователя сталкиваться с файловой системой, ставят успех в зависимость от понимания разницы между оперативной памятью и жестким диском.

Так термин «компьютерная образованность» становится эвфемизмом для социального и экономического апартеида и грубо разделяет наше общество на две части.

А как же те, кто не склонен пособничать технократам, не способен или не желает получать компьютерное образование? Такие люди, многие сознательно, а большинство в силу обстоятельств, остаются за бортом информационной революции. К примеру, многие высокотехнологические компании даже не рассматривают кандидатов, не имеющих адресов электронной почты. Уверен, есть множество подходящих по всем остальным параметрам кандидатов, которым не светит найм только потому, что они еще не подключены к Интернету. Что бы ни говорили апологеты, эффективно работать с электронной почтой сложно, это требует значительного уровня компьютерной образованности. Следовательно, речь идет об искусственной изоляции части рабочей силы. С моральной точки зрения это напоминает банковский прием «красная черта». Смысл этого незаконного приема в том, что все дома определенного района объявляются неприемлемыми в качестве обеспечения жилищных займов. Красные линии на карте предположительно проводятся по

экономическим контурам, но на деле слишком четко следуют контурам расовым. Банкиры заявляют, что никакого расизма здесь нет, однако результаты именно такие.

Говоря о «компьютерном образовании», программисты отделяют красной чертой и этнические группы, однако мало кто обращает на это внимание. Слишком сложно увидеть, что происходит в действительности, потому что картина замазывается технической мифологией. Легко увидеть, что банкир может выдать заем под любой дом. Но не очевидно, что программист может создавать интерактивные продукты, работа с которыми не вызовет затруднений у людей со слабой социоэкономической подготовкой.

Наша индустрия в целом отрицает существование проблемы пригодных к использованию интерактивных продуктов. Слишком много апологетов, ликующих по поводу танцующих медведей. Их театральные выкрики заглушают наши сомнения в работоспособности продуктов, основанных на программном обеспечении. Прежде чем начать поиск решений, мы все должны образумиться и оценить масштабы и остроту проблемы. Что мы и попытаемся сделать в следующей главе.

Часть II
Масштабные издержки

Глава 3
Пустая трата денег

Выбросить на ветер миллионы долларов не так легко, как кажется, однако некачественный процесс разработки – вполне подходящий инструмент для этой задачи. Дело в том, что в разработке программного обеспечения не хватает одного ключевого элемента: трудно понять, когда проект «готов». Не имея этого жизненно важного знания, мы слепо уповаем на произвольные сроки сдачи. Мы теряем миллионы в стремлении пересечь финишную черту как можно быстрее – лишь для того, чтобы обнаружить, что финишная черта оказалась миражом. В этой главе я попытаюсь развеять дорогостоящее заблуждение о возможности управления, ориентированного на фиксированные сроки сдачи.

Управление, ориентированное на крайние сроки сдачи

Некоторые странные традиции, принятые в Кремниевой долине, можно отнести на счет скорости выхода продукта на рынок. Часто предполагается, что *немедленный выпуск* продукта гораздо лучше, чем более поздний. Этот императив применяется в качестве оправдания предельно амбициозных сроков сдачи и нервного истощения сотрудников на работе. Следует скрыть более серьезные страхи. Сдача в три месяца продукта, раздражающего пользователей и приводящего их в ярость, совсем *не лучше* , чем сдача продукта, приятного для пользователей, в шесть месяцев – и всем деловым людям это прекрасно известно.

Причина одного из самых глубоких страхов руководителя в том, что он не знает, примет ли рынок продукт. Неспособность руководителя оценить завершенность продукта порождает другой страх. Если не принимать во внимание предельно ясные свойства вроде «работает на заданной конфигурации» и «не сбоит», руководители обычно не имеют четкого понимания состава законченного продукта.

Следствие этих двух страхов таково, что если программа «не сбоит», то не так уж и важно, как долго ее будут делать – три месяца или шесть, за исключением того, что в последнем случае стоимость разработки кошмарно увеличивается из-за лишних трех месяцев программирования. Когда программисты уже принялись за работу, деньги начинают таять очень быстро. Следовательно, логика подсказывает руководителю разработки, что самое

важное – как можно раньше начать и как можно раньше завершить написание кода.

Добросовестный руководитель быстро нанимает программистов и незамедлительно сажает их за работу. Он смело устанавливает дату завершения – через несколько месяцев после начала разработки, и команда, очертя голову, несется к финишной линии. Но если при этом продукт никто не проектирует, то два страха нашего руководителя остаются в силе. Он не смог узнать, понравится ли пользователям продукт, так что успешность продукта на рынке действительно остается тайной. Он также не установил, как должен выглядеть «завершенный» продукт, так что тайной остается и степень его завершенности. Позже я покажу, как проектирование взаимодействия способно исправить такое положение вещей. Сейчас же я продемонстрирую, насколько основательно фиксированные сроки сдачи подрывают процесс разработки, превращая неуверенность руководителя в неизбежно сбывающиеся предсказания.

Что такое «готово»?

Имея на руках конкретное описание завершенного программного продукта, мы можем сравнить наше творение с этим описанием и понять, готов ли продукт.

Существует два типа описаний. Мы можем создать подробное и полное вещественное описание продукта либо описать, какой реакции конечного пользователя на наш продукт необходимо добиться. Скажем, в традиционной архитектуре чертежи являются первым типом. Если же планируется создание фильма или нового ресторана, описание сосредотачивается на ощущениях, которые мы хотели бы передать своим клиентам. Для продуктов, основанных на программном обеспечении, обязательно использовать комбинацию обоих типов.

К сожалению, большинство программ *не имеет* точных описаний. Зато каждая характеризуется длинным перечнем функций, похожим на перечень ингредиентов. Магазинная корзина с мукой, сахаром, молоком и яйцами – совсем не то же самое, что пирог. Пирог получается лишь тогда, когда выполнены все инструкции рецепта, и результат выглядит как знакомый нам пирог, обладает таким же запахом и вкусом.

Обладая нужными ингредиентами, но не знанием о пирогах и выпечке, эрзац-повар будет впустую ковыряться на кухне безо всякой гарантии результата. Если мы потребуем, чтобы пирог был готов к шести часам, добросовестный повар, разумеется, принесет нам блюдо в назначенное время. Но будет ли эта стряпня пирогом? Мы знаем лишь, что продукт появится вовремя, а вот хорош ли он будет, остается тайной.

На традиционных строительных работах мы понимаем, когда работа завершена, потому что знаем, как выглядит «сделанная» работа. Мы знаем, что здание завершено, потому что выглядит и функционирует точно так, как задано в чертежах. Если срок сдачи объекта – первое июня, то наступление июня не всегда означает, что здание готово. Степень завершенности здания может быть определена лишь его сравнением с планом.

Без чертежей строители программ не могут четко осознавать, что делает продукт «завершенным», поэтому они выбирают возможную дату завершения и, когда она наступает, объявляют, что продукт готов. Уже первое июня, следовательно, продукт готов. «В тираж» – говорят они, и срок сдачи становится единственным признаком завершенности проекта.

Программисты и бизнесмены не дураки и не бестолочи, поэтому продукт не будет идти совершенно вразрез с реальностью. Он будет работать хорошо, без сбоев, и обладать достойным набором возможностей. Продукт будет работать достаточно хорошо в руках людей, *глубоко заинтересованных* в его хорошей работе. Не исключено даже, что продукт подвергался тестированию на практичность и удобство применения (юзабилити-тестированию). При этом посторонних людей просили поработать с продуктом под внимательным наблюдением специалистов по юзабилити (Usability Professionals)[8]. Но

[8] Специалист по юзабилити (usability professional), или эргономист, и проектировщик взаимодействия (interaction designer) – разные люди. Подробно о различиях рассказано в главе 12 «В отчаянных поисках

даже будучи весьма разумными, эти предосторожности не позволяют ответить на фундаментальный вопрос: станет ли продукт успешным?

Закон Паркинсона

Руководителям известно, что разработка программного обеспечения подчиняется закону Паркинсона: работа увеличивается в объеме, занимая любое отведенное под нее время. Если вы заняты в бизнесе программного обеспечения, то, вероятно, знакомы со следствием закона Паркинсона, известным в качестве правила Девяносто-Девяносто (авторство приписывается Тому Каргилу из Bell Labs):

> «Первые 90% кода отнимают первые 90% времени разработки. Оставшиеся 10% кода отнимают вторые 90% времени разработки».

Иными словами, это самоуничижительное правило гласит, что когда программисты написали 90% кода, они *все еще* не знают, где находятся! Руководство отлично понимает, что успеть сдать работу вовремя нельзя, какие сроки сдачи ни устанавливай. Разработчики же лучше всего работают под давлением, поэтому руководство использует дату сдачи, как одно из средств давления.

В восьмидесятые и девяностые годы Ройял Фаррос был вице-президентом небольшой, но влиятельной компании T/Maker, где руководил разработкой. Вот его слова:

> «Многие из нас устанавливали сроки сдачи *заведомо* невозможные, причем настолько, что делали верным одно из следствий закона Паркинсона: «Для завершения проекта требуется в два раза больше времени, чем было отведено». Я *твердо* верил, что если выделить под проект шесть месяцев, он займет год. Так что если необходимо было получить что-то через два года, следовало требовать сдачи через год. Тупое принуждение, конечно, но оно всегда срабатывало».

Когда предприниматель от программного обеспечения Риджели Эверс работал в Intuit над созданием QuickBooks, он столкнулся с той же проблемой.

> «Выпуск первой версии QuickBooks должен был занять девять месяцев. Мы правильно оценили, что начальное развитие будет длиться столько же, сколько длится беременность, но все-таки ошиблись: у нас ушло почти два с половиной года – срок беременности слона».

Архитектор программного обеспечения Скотт Мак-Грегор указывает, что закон Грешема – плохая валюта вытесняет хорошую – также имеет силу в этом контексте. Когда на рынке сосуществует пара валют, люди создают запасы сильной валюты и стараются тратить слабую. В конечном итоге это приводит к преобладанию слабой валюты. Так и некачественные оценки завершения проекта вытесняют реальные прогнозы. Когда все делают радужные, но взятые с потолка прогнозы, руководитель, выдающий реалистические и более долгие сроки, производит впечатление, будто специально занимается саботажем. Такой руководитель подвергнется давлению, будет вынужден занизить свои оценки.

Сроки сдачи в некоторых проектах неразумны по причине произвольности. Рациональные руководители в большинстве своем по-прежнему склонны устанавливать сроки хотя и физически возможные, но возможные лишь ценой больших жертв. Пилот самолета, по аналогии, может заявить: «Успеем в Чикаго вовремя, но багаж придется выбросить!» Мне приходилось видеть, как руководители разработки приносят в жертву не только проектирование, но и тестирование, применимость, функции, интеграцию, документацию и даже реальность. *Большинство руководителей разработки продуктов, с* эргономики».

которыми мне приходилось работать, предпочтут выбросить на рынок неработоспособный продукт, но не опоздать со сдачей этого продукта.

Продукт, вечно не готовый к выпуску

Часто причиной этому служит глубочайший страх каждого, кто руководит разработкой программного обеспечения: если продукт опаздывает, он вообще никогда не будет выпущен. Нет сомнения в правдивости историй о продуктах, которым так и не суждено было увидеть свет. Проект опаздывает сначала на год, потом на два, а потом, на третьем году жизни, его мстительно подвергают эвтаназии руководители высшего звена или члены директората. Это объясняет неистовую приверженность к фиксированным срокам сдачи, пусть даже ценой жизнеспособности продукта.

Для примера – в конце девяностых годов в широко разрекламированной начинающей компании Worlds, Inc. масса умных и способных людей работали над созданием сетевого виртуального мира, где люди, посредством своих аватар, могли бы путешествовать и общаться с другими людьми в реальном времени. Продукт никогда не имел четкого определения или описания, и после растраты десятков миллионов инвестированных долларов члены правления компании милосердно прекратили эту агонию.

В начале девяностых другая начинающая компания, Nomadic Computing, потратила около 15 миллионов долларов, пытаясь создать новый продукт для мобильных деловых людей. К сожалению, никто в этой компании не знал точно, что это за продукт. Все было ясно с рыночной нишей продукта, и были известны возможности программы, однако не было четкого понимания целей потенциальных пользователей. Словно безумные скульпторы, обтесывающие гигантский мраморный камень в надежде обнаружить внутри статую, разработчики писали невероятные объемы бесполезного кода, который в конечном итоге просто выбросили, вместе с деньгами, временем, репутацией и карьерами. Самая же печальная потеря в этой истории – потеря возможности создать востребованную программу.

Даже корпорация Microsoft не обладает иммунитетом против подобных заблуждений. Ее первая попытка создать продукт для управления базами данных в конце восьмидесятых годов поглотила множество человеко-лет, и Билл Гейтс в конечном итоге, милосердно закрыл проект. Преждевременная смерть проекта ударной волной прокатилась по сообществу разработчиков. Следующий продукт этого направления, Access, разрабатывался с нуля другими разработчиками и другими руководителями.

Поздний выпуск – не беда

По иронии судьбы поздний выпуск продукта обычно не является смертельным событием. Опоздавший продукт среднего качества часто оказывается провальным, но если ваш продукт представляет ценность для пользователей, нарушение расписания не обязательно приведет к долговременным отрицательным эффектам. Если продукт оказывается настоящим хитом, то опоздание на месяц – и даже на год – значения совершенно не имеет. Напротив, если продукт отвратительный, – кому интересно, что он выпущен вовремя?

Конечно, для отдельных массовых продуктов, приносящих основную прибыль только в сезон рождественских подарков, сроки могут быть ужасно важны. Но продукты, основанные на программном обеспечении, даже если они потребительские, не настолько чувствительны к датам.

К примеру, созданный в 1990 году компанией GO компьютер Penpoint должен был стать прародителем современных карманных и наладонных компьютеров. В 1992 году, когда Penpoint потерпел крах, компьютер Apple Newton перехватил флаг революции КПК. Когда и Newton не смог привлечь людей, новой надеждой в этом направлении стал компьютер Magic Link от General Magic. Это было в 1994 году. Когда продажи Magic Link провалились, рынок

КПК словно умер. Инвесторы объявили его бесперспективным. Затем, как гром среди ясного неба, в 1996 году к славе и успеху вознесся PalmPilot. Он захватил невспаханную целину рынка, *опоздав на шесть лет.* Рынок всегда готов принимать хорошие продукты, имеющие ценность и привлекательность для пользователей.

Разумеется, компании, зарекомендовавшие себя в создании аппаратных средств, сегодня создают гибридные варианты, содержащие микросхемы и программный код. Они склонны недооценивать влияние программного обеспечения и пытаются подчинить эту область рутинным процедурам, созданным для разработки аппаратного обеспечения. Что в корне неверно, потому что, как показано в главе 1 «Загадки века информации», компании эти теперь работают в сфере программного обеспечения (даже если их сотрудники об этом не знают).

Торг за набор функций

Одним из последствий управления, ориентированного на фиксированные сроки сдачи, является феномен, который я называю «торгом за набор функций».

Много лет назад программисты записывали спецификации продукта (в весьма произвольной форме) на салфетках во время вечеринок и пожалели об этом, потому что именно их обвиняли в столь частом появлении и неудачных программ. В целях самозащиты программисты потребовали, чтобы руководители и маркетологи точнее излагали свои требования. Компьютерные программы основаны на процедурах, а процедуры легко преобразуются в возможности, поэтому было вполне естественно, что для программистов «точность» означала конкретный список возможностей. Этот перечень функций программ позволял программистам переносить вину на руководство, когда продукт не оправдывал надежд. Они всегда могли сказать: «Мы ни при чем. Мы реализовали все возможности, перечисленные руководством».

В результате большинство продуктов начинает свое существование с документа, который в разных случаях называется техническим описанием или маркетинговыми требованиями. Проще говоря, это перечень желательных возможностей – вроде перечня ингредиентов в рецепте пирога. Такой документ обычно является результатом ряда длительных мозговых штурмов, где руководители, маркетологи и разработчики придумывают отличные возможности и кратко фиксируют их. Излюбленный инструмент для создания таких списков – электронные таблицы, и типичная таблица может занимать десятки страниц. (По традиции одна из строчек, конечно же, содержит слова «хороший пользовательский интерфейс».) Предложения, касающиеся возможностей продукта, также исходят от фокус-групп, добавляются по результатам рыночных исследований и анализа продуктов конкурентов.

Руководители передают набор функций программистам и говорят: «Продукт должен быть выпущен к первому июня». Программисты, разумеется, соглашаются, но с некоторыми оговорками. Перечисленных функций слишком много, чтобы успеть реализовать все в отведенное время, говорят они, и многие возможности придется урезать, чтобы успеть к сроку. Так начинается освященный веками процесс торга.

Программисты проводят черту ровно посередине списка. Возможности над чертой будут реализованы, провозглашают они, а возможности за «линией смерти» – отложены на потом или вовсе вычеркнуты. Руководство стоит перед выбором: увеличить время разработки или урезать функциональность. Проект неизбежно займет больше времени, чем запланировано, но руководство ненавидит признавать этот факт в начале проекта, поэтому начинается торг за функции. Здесь хватает и споров и цирковых представлений. Возможностями жертвуют за время, временем – за возможности. Эти примитивные капиталистические переговоры настолько присущи природе человека, что обе стороны чувствуют себя очень неплохо. Здесь появляются изощренные параллельные стратегии; как указывает Ройял Фаррос (Royal Farros) из T/Maker,

> «если одну из функций обвинили в задержке сроков сдачи, это позволяет десятку других замедляющих функций пробраться в перечень безо всяких последствий».

В этом сражении теряется перспектива, необходимая для успеха.

Фаррос описывает флагманский продукт компании T/Maker, текстовый процессор WriteNow, как

> «идеальный продукт для университетской среды. В 1987 году мы продали в этом сегменте больше копий WriteNow, чем Microsoft продала копий Word. Однако мы не смогли удерживать лидерство, потому что рассердили своих самых преданных поклонников на этом рынке, не добавив самую нужную для них функцию: *концевые сноски.* Пытаясь успеть к сроку, мы не смогли включить ее в спецификацию. Мы успели к сроку, но потеряли целый сегмент рынка».

Кто главный? Программисты

Несмотря на внешние признаки, программисты полностью контролируют этот процесс принятия решений снизу вверх. Именно они определяют, сколько времени займет реализация каждой возможности, и поэтому могут перемещать требования в конец списка, просто посчитав их трудоемкими. Программисты, из соображений самозащиты, назначают нечетко определенным позициям большую трудоемкость, причем, как правило, речь идет о существенных вопросах взаимодействия с пользователем. Это неизбежно перемещает вопросы пользовательского интерфейса в конец списка. Наверх же всплывают более привычные идиомы – простые в создании меню, мастеры, диалоги. Анализ и скрупулезные размышления, проведенные обладающими властью и высокими зарплатами исполнительными лицами, в одностороннем порядке превращаются в спорные идеи программистом, имеющим собственные соображения или защищающим собственную территорию.

Руководители попадают в незавидное положение и могут влиять лишь на незначимые параметры процесса разработки, словно пытаясь увеличить громкость колонок, в любом случае находящихся вне зоны слышимости. Нет сомнения, что руководству необходимо контролировать процесс создания и выпуска успешных программных продуктов, но, к сожалению, наш культ фиксированных сроков сдачи полностью игнорирует критерии «успешности», сосредотачиваясь на факте «создания». Мы вручаем создателям продукта бразды правления, переводя таким образом руководителей на роль пассажиров и наблюдателей.

Возможности не всегда нужны

Несмотря на кажущееся пристрастие к функциональности, пользователи не слишком стремятся получить максимум возможностей. Успехи и неудачи продуктов неоднократно демонстрировали, что пользователей не очень волнуют функции продуктов. Они интересуются лишь возможностью решать задачи. Иногда функции необходимы для этого, но чаще они просто смущают пользователей и мешают им делать работу. Бесполезные возможности заставляют пользователей чувствовать себя глупо. Если обратиться к предыдущему примеру, успешный PalmPilot обладал гораздо меньшим количеством функций, чем провалившиеся Magic Link от General Magic, Newtown от Apple и компьютер Penpoint. Своим успехом PalmPilot обязан проектировщикам, единодушно сосредоточившимся на целевой аудитории и ее потребностях.

Что я могу сказать хорошего о функциях? Они поддаются измерению. И это свойство

измеримости придает им ауру ценности, которой в действительности они не обладают. Отрицательные качества функций полностью съедают все их положительные качества. Функции – причина одной из серьезных проблем проектирования, потому что каждая возможность, предложенная из лучших побуждений и *потенциально* полезная, оттеняет некоторые возможности, которые, вероятно, будут *действительно* полезны. Разумеется, реализация возможностей не бесплатна. Каждая из них усложняет продукт. Они требуют увеличения размера и сложности документации и системы контекстной справки. Что еще важнее, с точки зрения затрат они требуют раздувания штата технической поддержки, занятого в консультировании пользователей относительно этих самых возможностей.

Для нашего зациклившегося на функциях мира мысль, наверное, непривычная – вы не достигнете своих целей, используя набор функций, как инструмент. Можно замечательно реализовать все функции из утвержденного набора и все равно попасть в беду. Для доказательства этого тезиса проектировщик взаимодействий Скотт Мак-Грегор на своих занятиях использует вот такой замечательный тест. Он описывает продукт с помощью перечня функций и просит слушателей записать, что это за продукт, как только они догадаются. Он перечисляет: 1) двигатель внутреннего сгорания; 2) четыре колеса с резиновыми покрышками; 3) трансмиссия, связывающая двигатель с ведущими колесами; 4) трансмиссия и двигатель смонтированы на ходовой части; 5) рулевое колесо. На этот момент времени каждый слушатель уже записал, что это автомобиль, но здесь Скотт перестает описывать особенности продукта и вместо этого называет пару задач потенциального пользователя: 6) быстро и легко срезает траву; 7) на этом удобно сидеть. На основании пяти функций-подсказок ни один слушатель не может догадаться, что это минитрактор-газонокосилка. Очевидно, что цели пользователя намного более наглядны, чем набор функций продукта.

Итерации и миф о непредсказуемости рынка

В отрасли, столь переполненной деньгами и возможностями их заработать, часто бывает проще начать новое предприятие и списать последнюю неудачу на случайности, чем отнести ее на счет *реальной* причины.

В начале девяностых мне пришлось оказаться участником одного из таких провалов. Я способствовал созданию компании с инвестиционным финансированием. Заявленной целью компании было предельное упрощение задачи объединения персональных компьютеров в сети[9]. Продукт хорошо работал, был прост в применении, но из-за печальной серии грубых маркетинговых ошибок провалился, как ни прискорбно. Не так давно, будучи на конференции, я столкнулся с одним из инвесторов, входившим в правление той злополучной компании. Мы не общались со времен того провала и, словно ветераны, потерпевшие совместно поражение на реальном поле боя много лет назад, утешили друг друга, как умудренные опытом люди. Но, к моему невероятному удивлению, этот в других отношениях крайне успешный и умный человек поделился своим откровением: несмотря на безупречность усилий в области маркетинга, руководства и технической подоплеки, потенциальные покупатели «просто не были заинтересованы в легком разворачивании локальных сетей». Столь очевидно глупое заявление меня ошеломило, и я возразил, что, несомненно, потребность в таком решении существовала, и виновата была лишь наша неспособность обеспечить такое решение на должном уровне. Он переформулировал свое заявление, приведя убедительные аргументы в пользу того, что мы попросту продемонстрировали отсутствие у людей потребности в простых сетях.

Позже тем вечером, пересказывая эту историю супруге, я осознал, что его обоснование

[9] Мы говорили, что хотим сделать «объединение в сети компьютеров Intel/Windows столь же простым, как объединение в сети компьютеров Macintosh». В то время Mac’и до смешного просто объединялись в сети посредством протокола AppleTalk. Тогда, как и сегодня, объединять ПК Wintel в сети было трудно.

провала определенно было удобным для всех участников того проекта. Свалив неудачу на случайную флуктуацию рынка, мой коллега оправдал инвесторов, руководителей, маркетологов и разработчиков, освободил их от вины. И реальность такова, что каждый из участников той компании позже нашел себя в других успешных начинаниях в Кремниевой Долине. У этого инвестора теперь надежный инвестиционный портфель в других успешных компаниях.

В процессе разработки того сетевого продукта все его функции записывались в перечень возможностей. Проект уложился в бюджет. Продукт появился в срок (в действительности мы постоянно оттягивали выход, но в определенный срок продукт все-таки появился). Все поддающиеся количественному измерению аспекты разработки находились в пределах нормы. Единственное заключение, которое мог сделать этот сведущий в руководстве инвестор, сводилось к существованию неожиданной аномалии рынка. Как мы могли потерпеть неудачу, если все датчики показывали нормальные цифры?

Объективность подобных измерений придает всем уверенности. Объективные и количественные показатели пользуются высоким уважением среди программистов и деловых людей. Неэффективность же таких измерений с точки зрения создания успешных продуктов как-то упускается из виду. Если продукт оказывается успешным, отцы-основатели приписывают все заслуги себе, относя победу на счет своего замечательного понимания технологии и рынка.

С другой стороны, если продукт терпит неудачу, ни у кого нет ни малейшего стимула выкапывать останки и анализировать эту неудачу. Сойдет любое оправдание, если у игроков – руководителей и разработчиков есть возможность перейти к следующей высокотехнологичной идее, коих существует неприлично много. Таким образом, нет причин убиваться из-за неудач. Неприятный побочный эффект непонимания неудач состоит в том, что молчаливо признается невозможность предсказания успеха, все считают, что удача и случай правят миром высоких технологий. Это обстоятельство, в свою очередь, работает за метод финансирования, известный среди инвесторов как «распыли и молись». Деньги при этом вкладываются небольшими частями в многочисленные предприятия в расчете на то, что одно из них окажется успешным.

* * *

Среды быстрой разработки, такие как Всемирная паутина и Visual Basic до нее, также внесли свой вклад в развитие подхода, согласно которому повторять попытки следует до тех пор, пока одна из них не окажется успешной. Всемирная паутина, будучи новой рекламной площадкой, привлекала и привлекает массу специалистов по маркетингу, которые особенно восприимчивы к мифу о непредсказуемости рынка и, следовательно, правилу повторения попыток. Маркетологам хорошо знаком жесткий и капризный мир рекламы и средств массовой информации. В конце концов, реклама во многих случаях – *просто стрельба наугад.* Например, в рекламе «новое» считается наиболее эффективной концепцией для продвижения продуктов на рынке, однако когда компания Coca-Cola десять лет назад представила миру новый напиток «New Coke», она потерпела фиаско. Никто не смог бы предсказать такой исход. Вкусы и предпочтения людей меняются случайным образом, поэтому может казаться, что эффективность маркетинга тоже зависит от случая.

Во Всемирной паутине эта проблема возникает, когда веб-сайт из сетевого каталога вырастает в сетевой магазин. Из простой презентации превращается в интерактивный программный продукт. Люди из СМИ и рекламы, достигшие значительного успеха с первым поколением сайтов, теперь пытаются применить все те же итеративные методы для интерактивного сайта и попадают в беду, часто даже не осознавая этого. Результаты маркетинга могут быть случайными, а результаты взаимодействия – нет. Именно когнитивное сопротивление, порождаемое интерактивностью программного обеспечения, производит на неподготовленного пользователя впечатление хаоса.

Невероятная гибкость Интернета способствует подобному положению дел, поскольку проведение рекламных или маркетинговых компаний в данном случае требует кардинально меньших затрат денег и времени, в отличие от печатной или телевизионной рекламы. Специалист по вебмаркетингу может получать практически мгновенный отклик, оценивая эффективность рекламы, вследствие чего скорость прохождения циклов возрастает неимоверно, и иногда серьезные изменения вносятся в течение одного дня. На практике получается «на авось». Многие руководители начинающих веб-компаний действительно применяют удивительно простой принцип проектирования наудачу. Они создают клон какой-нибудь древней программы, разработка которого почти не требует затрат времени, и демонстрируют результат пользователям. Затем они выслушивают жалобы и отзывы, оценивают сценарии использования программы, дорабатывают слабые места и снова выпускают программу на рынок. Программистам, как правило, итеративный метод не слишком нравится, поскольку увеличивает объем их работы. Но итеративный процесс обычно нравится именно руководителям, слабо знакомым с технологией, поскольку этот процесс освобождает их от необходимости тщательного планирования, от размышлений и от необходимости усердствовать (иными словами, от проектирования взаимодействий). И конечно, дороже всего за подобное отношение платят пользователи. Им приходится продираться через одну за другой вялые попытки авторов, пока они не получат наконец программу, сносную в использовании.

Исходя лишь из того, что отзывы покупателей улучшают ваше понимание продукта или услуги, нельзя делать вывод, что метод случайного добавления функций и последующей реакции на положительные или отрицательные отзывы клиентов – эффективный, дешевый или даже просто действенный. В мире танцующих медведей подобная стратегия жизнеспособна в минимальной степени, а на любом рынке, проявляющем хоть малейшие признаки конкуренции, она самоубийственно глупа. Даже если на рынке больше никого нет, это весьма расточительный метод.

Многие руководители, восприимчивые и профессиональные в других отношениях, совершенно бесстыдно гордятся этим методом. Один зрелый и опытный руководящий работник (в прошлом маркетолог) однажды спросил меня в тоне самозабвенной риторики: «Как может кто-то утверждать, что знает, чего хочет пользователь?» Поразительный вопрос. Каждый бизнесмен полагает, что знает это. Ценность большинства деловых людей как раз в их «предположениях» относительно потребностей покупателя. Да, такие предположения наверняка разойдутся с потребностями некоторых пользователей, но отсутствие предположений означает, что результат не понравится *никому*. Тот глупец считал, что его клиенты согласны преодолевать последствия его новых предположений, выполняя за него работу по проектированию. Возможно, что сегодня в Кремниевой Долине нашлось бы немало путешествующих по Паутине апологетов, готовых помочь этому лентяю разобраться с его бизнесом, но при этом скольких *уцелевших* он отпугнул своим надменным отношением? Пока он переделывал работу, реагируя лишь на тех, у кого хватило выдержки вернуться на его веб-сайт еще раз, скольких клиентов он потерял навсегда? Чего хотели они? Говорят, Сталин расчищал минные поля, посылая на них штрафные батальоны. Эффективно такое решение? Да. Рационально, гуманно, жизнеспособно, привлекательно? Нет.

Конечно, самый серьезный недостаток метода в том, что он изначально отпугивает всех уцелевших и остаются лишь пользователи-апологеты. Это существенно искажает природу и качество отзывов, ограничивая аудиторию апологетами-технофилами, то есть очень небольшой долей рынка. Именно по этой причине очень немногие программные продукты для персональных компьютеров успешно становятся массовыми.

Я вовсе не хочу сказать, что нельзя учиться методом проб и ошибок, однако эти пробы должны основываться на чем-то большем, чем слепой случай, они должны начинаться с хорошо продуманного решения, а не тяп-ляпа за один вечер. В противном случае ленивый бизнесмен всегда имеет оправдание своего некорректного обращения с клиентами.

Скрытые издержки некачественного программного обеспечения

Если программа раздражает пользователей и сложна в применении, люди станут избегать работы с ней. Не слишком примечательный факт, если не осознать, что работа многих людей связана с применением программ. Корпоративные затраты на использование таких программ невозможно измерить, однако они вполне ощутимы. Как правило, эти затраты выражаются не в деньгах, но в других более критических валютах, таких как время, уровень беспорядка, репутация и преданность клиентов.

Пользователи делового программного обеспечения могут презирать его сколько угодно, но им платят за то, чтобы они терпели эти программы. В результате изменяется восприятие людьми программ. Пользователям платят за работу с программным обеспечением, поэтому они становятся гораздо более терпеливыми к его недостаткам – ведь у них нет выбора, однако применение подобных программ не становится из-за этого дешевле. Напротив, затраты остаются высокими, а заметить и учесть их становится очень трудно.

Некачественно спроектированные бизнес-приложения вызывают у людей неприязнь к работе. Производительность страдает, в работе появляются ошибки, начинается борьба с программами, возрастает текучесть кадров. Потеря сотрудника стоит очень дорого, причем не только в финансовом выражении, но еще и в нарушении деятельности предприятия: потерянное время никогда не вернуть. Многие из тех, кто получает деньги за применение определенных инструментов, испытывают стеснение из-за того, что не могут на эти инструменты жаловаться, однако это не мешает им раздражаться и быть недовольными по этому поводу.

Одна из самых затратных статей, связанных со сложными в применении программами, – это техническая поддержка. Microsoft ежегодно тратит 800 миллионов долларов на техническую поддержку. А ведь речь идет о компании, которая тратит многие сотни миллионов долларов на юзабилити-тестирование и исследования. Очевидно компания Microsoft убеждена, что такие масштабы поддержки – неизбежное зло. Я в это не верю. Представьте, какие преимущества получит ваша компания, если вы не будете так думать. Представьте, насколько более эффективными станут ваши усилия по разработке, если вы сможете сохранить пять процентов прибыли, не оплачивая техническую поддержку.

Спросите любого, кому пришлось поработать в службе технической поддержки любой компании, создающей приложения для настольных компьютеров, и этот человек скажет, что большая часть его времени и усилий уходит на разъяснение вопросов, связанных с файловой системой. Совсем как Джейн из главы 1, пользователи не понимают рекурсивную иерархию файловой системы – будь то Finder или Explorer, система Windows, Mac или UNIX. Как ни странно, очень немногие компании тратят средства на проектирование и реализацию более дружественных к человеку альтернатив файловой системе. Все прочие выбирают гораздо более дорогой вариант бесконечной телефонной поддержки по связанным с файловой системой вопросам.

Можете винить «глупого пользователя» сколько хотите, однако вам все равно придется нанимать дорогостоящих сотрудников в службу технической поддержки, если вы собираетесь продавать и распространять программы, не спроектированные как следует.

Дороже разработки ПО обходится только разработка плохого ПО

Программисты стоят дорого, а программисты, сидящие без дела в ожидании завершения проектирования, крайне раздражают руководителей. Глупо же, думает руководитель, что программисты сидят и ждут, хотя могли бы программировать. Заставить программистов работать до завершения этапа проектирования – ложная экономия. Когда появляется программный код, процесс уже не остановить, поэтому проектировщики вынуждены реагировать на потребности программистов, а должно быть наоборот. И вправду,

глупо заставлять своих программистов ждать, а ведь очень просто сделать так, чтобы они не сидели без дела – надо, чтобы проектировщики взаимодействия планировали следующий продукт или релиз параллельно с созданием текущего продукта или релиза.

В долгосрочной перспективе беспорядочное программирование обойдется дороже, чем полное отсутствие программирования. Эта истина настолько противоречит здравому смыслу, что большинство руководителей никак ее не воспринимает. Когда код написан, очень трудно его выбросить. Подобно писателям, влюбленным в свою прозу, программисты привязываются к своим алгоритмам на эмоциональном уровне. Модификация программы на полуслове вносит беспорядочность в процесс разработки и вредит коду. Руководителю еще труднее выбросить код, потому что он дорого заплатил за его создание и хорошо понимает, что замена обойдется еще дороже.

Если проектирование не предшествует программированию, вряд ли оно окажет какое-либо влияние. Один руководитель сказал мне: «Наши люди уже пишут код, и я не собираюсь их останавливать». Эти ковбои думают: «Пока мы будем лететь к земле, я успею сшить парашют». Отважное заявление, однако, мне не довелось видеть ему подтверждения.

Не имея результатов серьезного этапа проектирования, программисты непрерывно экспериментируют со своими программами в поисках лучших решений. Они действуют так же расточительно, как плотник, распиливающий доски «на глаз», пока не зашьет дыру в стене.

Свойства неизмеримости и неосязаемости программного обеспечения препятствуют точной оценке его масштабов и завершенности. Добавьте любовь программиста к своему ремеслу и вы поймете, что проекты неизбежно распухают в объеме и времени. Программируя подобным образом, мы всегда будем получать сюрпризы, пока не начнем правильно устанавливать промежуточные сроки и определять, где мы находимся.

Стоимость возможностей

В эпоху информации дороже всего обходится не создание чего-либо, а потерянная возможность создать это. Создание провального продукта означает, что вы не создали успешный. Если для создания хорошего продукта потребовалось в течение трех лет выпускать по одной его версии, значит, за три года вы не создали три хороших продукта. Основной бизнес компании Novell – сети, но она же пыталась открыто состязаться с Microsoft в области офисных приложений. Попытки пробиться на этот рынок обошлись Novell очень и очень недешево, однако самой серьезной потерей стала потеря лидерства на сетевом рынке. Деньги – ничто в сравнении с исключительной возможностью момента.

Компания Netscape утратила лидирующие позиции на рынке броузеров точно таким же образом, а именно когда решила состязаться с Microsoft в сегменте операционных систем.

Каждый разработчик продуктов, основанных на кремнии, должен оценить и изучить самые важные цели своих пользователей и стойко сосредоточиться на достижении этих целей. Слишком уж легко поддаться соблазну многочисленных возможностей в области высоких технологий и упустить главный шанс. Программисты, независимо от их интеллекта, деловой хватки, преданности и добрых намерений, прислушиваются к несколько иным мотивам и способны легко отвлечь предприятие от правильных целей.

Издержки прототипирования

Прототипирование – это то же программирование, с той же основой и затратами, однако результат не обладает эластичностью настоящего кода. Программные прототипы – это строительные леса, они имеют мало общего с долгоживущим, расширяемым кодом, пригодным к сопровождению, эквивалентом каменных стен. Руководители, в частности, неохотно выбрасывают работающий код, даже если это прототип. Они не видят разницы между строительными лесами и каменными стенами.

Прототип можно создать гораздо быстрее, чем настоящую программу. Что и делает прототип привлекательным, ведь он кажется столь недорогим; однако, программирование дает надежную программу, тогда как создание прототипа дает лишь шаткий фундамент. Прототипы – это эксперименты, результаты которых надлежит выбрасывать, хотя в реальной жизни прототипы чаще сохраняют. Руководители смотрят на работающий прототип и спрашивают: «Почему бы нам просто не использовать это?» Ответ технически слишком сложен и перегружен неопределенностью, чтобы переубедить руководителя, который видит перед собой возможность сэкономить многие месяцы дорогостоящих усилий.

Суть хорошего программирования в отсроченном вознаграждении. Выкладываясь в начале, вы пожинаете плоды позже. Немного найдется задач, выполнение которых вручную обойдется дороже. Однако единожды написанную программу можно выполнять миллионы раз, не неся дополнительных затрат. Самая дорогая программа – та, что будет запущена только один раз. Самая дешевая – та, что будет запущена десять миллиардов раз. Если не принимать во внимание крошечные программы, типа тех, что пишутся в школьные годы, экономика программного обеспечения странным образом полностью видоизменилась: самые дешевые программы с точки зрения пользователей дороже всего в разработке, а самые дорогие для пользователя, наоборот, дешевле в разработке.

Создание большой программы можно сравнить с постройкой столба из кирпича. Этот столб состоит из тысячи кирпичей, положенных один на другой. Столб может быть выстроен, только если класть кирпичи с большой точностью. Любое отклонение приведет к падению кирпичей. Если кирпич номер 998 отклонится на пять миллиметров, столб, вероятно, сможет выдержать тысячу кирпичей, но если отклонение в пятом кирпиче, столб никогда не станет выше трех десятков.

Это характерная особенность программного обеспечения – фундамент намного чувствительнее к манипуляциям, чем программный код более высоких уровней. В процессе конструирования любой программы разработчик совершает ошибки и вносит изменения по ходу действий. Как следствие, программа обрастает рубцовой тканью измененного кода. В любой программе существуют рудиментарные функции и нереализованные возможности. В каждой программе существуют возможности и процедуры, потребность в которых обнаружилась через какое-то время после начала работы. Каждый из этих шрамов – маленькое отклонение на вертикали кирпичей. Перенос кнопки с одной стороны диалога на другую эквивалентен подталкивание кирпича с номером 998, а изменение кода, отвечающего за отображение всех кнопок, – подталкивание пятого кирпича. Объектно-ориентированное программирование и принципы инкапсуляции данных – эта защитные методы, единственное назначение которых в том, чтобы защитить программу от образования рубцовой ткани. По сути дела, объектно-ориентированный подход разделяет башню из 1000 кирпичей на десять башен по 100 кирпичей.

Хорошие программисты тратят невероятное количество времени и сил при подготовке к созданию большой программы. Настройка среды программирования может продлиться несколько дней, прежде чем будет написана хотя бы строка кода будущего продукта. Необходимо отобрать подходящие библиотеки. Определить структуры данных. Проанализировать подсистемы хранения и поиска данных, определить их, закодировать и протестировать.

Углубляясь в работу, программисты неизбежно обнаруживают ошибки планирования и изъяны своих предположений. Они сталкиваются с гобсоновским выбором – потратить время и силы на то, чтобы исправить все, с самого начала, или же решить проблему на месте, создав новый рубец в виде отклонения от плана. Давать задний ход всегда очень дорого, однако рубцовая ткань в конечном итоге ограничивает размер программы – высоту кирпичной вертикали.

При каждом изменении программы – будь то исправление ошибок или добавление функций – появляются новые рубцы. Именно поэтому программы следует выбрасывать и полностью переписывать каждые пару десятков лет. Рубцовая ткань с течением времени

становится настолько толстой, что препятствует нормальной работе.

Прототипы по своей природе представляют собой программы, создаваемые в спешке и позволяющие проверить некоторые предположения. Чтобы быстро создать прототип, программист должен пожертвовать идеальным выравниванием кирпичей. Здесь не используются «правильные» структуры данных, информация бессистемна. Здесь не используются «правильные» алгоритмы, но используются любые подвернувшиеся фрагменты кода. Прототип *начинает* существование как масса рубцовой ткани. Он не может вырасти очень большим.

Некоторые разработчики пришли к прискорбному выводу, что современные инструменты для быстрого создания прототипов, такие как Visual Basic, представляют собой эффективные инструменты проектирования. Вместо того, чтобы заняться проектированием продукта, они наскоро создают крайне бледную версию продукта при помощи инструмента визуального программирования. Этот прототип, как правило, становится фундаментам продукта. Ради иллюзорных выгод в жертву приносится надежность продукта и продолжительность его жизни. Карандаш, лист бумаги и хорошая методология позволяют лучше спроектировать продукт, чем любое количество прототипов.

Для людей, не являющихся проектировщиками, визуализация формы еще не существующей программы затруднительна, а часто и невозможна. Для этих деловых людей прототипы исполняют роль инструмента визуализации. Поскольку прототип – это приближенная модель, созданная на основе существующих и доступных на момент разработки инструментов, он по определению полон временных компромиссов. Однако работающая программа, независимо от того, насколько плохо она работает, производит мощнейшее впечатление на тех, кому придется платить за ее разработку. Движущийся, пусть и хромающий, прототип обладает опасной силой, не соответствующей своей действительной ценности.

Силен соблазн руководителя сказать: «Не выбрасывайте прототип, используем его как фундамент *настоящего* продукта». Такое решение часто, в конечном итоге, препятствует появлению продукта на рынке. Программисты оказываются приговоренными к постоянной реанимации программы, дающей по мере своего развития смертоносные сбои. Это башня из кирпичей, где первые 25 кирпичей положили наудачу: независимо от того, насколько точно вы кладете все последующие кирпичи, насколько тщательно работает каменщик, насколько крепко держит строительный раствор, сила гравитации неизбежно разрушит башню где-то на пятидесятом уровне.

Ценность прототипа в знаниях, приобретенных в процессе его создания, а совсем не в коде. Мудрый разработчик Фредерик Брукс говорит: «Планируйте выбросить одну версию». Так или иначе, вы ее выбросите, так почему не запланировать это событие с самого начала?

В 1988 году я продал Биллу Гейтсу программу под названием Ruby, представлявшую собой язык визуального программирования, который в сочетании с продуктом Билла QuickBasic стал средой Visual Basic. Ruby была просто прототипом, но демонстрировала некоторые значительные новшества в подходе и технологии (при первом знакомстве с Ruby Билл спросил: «Как ты это сделал?»). Проектом Ruby стал заниматься тогдашний руководитель разработки Windows 3.0 Расс Вернер. По договору я должен был завершить разработку Ruby и представить полноценный продукт. Первое, что я сделал, – выбросил прототип и начал все с нуля, не имея ничего, кроме знаний и опыта. Когда Расс узнал об этом, он пришел в изумление, ярость и негодование. Он никогда не слышал о столь возмутительных действиях и выражал убежденность, что отказ от прототипа задержит выпуск продукта. Но это был уже свершившийся факт, и, несмотря на страхи Расса, мы сдали золотую мастер-копию в срок. После интеграции с языком Basic запуск среды VB стал одним из самых успешных для Microsoft. Windows 3.0, напротив, задержалась более чем на год и впоследствии пользовалась дурной славой из-за большого объема рудиментарного кода, унаследованного из прототипа.

В целом, далекие от технологии руководители ошибочно ценят завершенный код,

независимо от его надежности, гораздо выше, чем замысел, или мнение тех, кто этот код писал. Коллега Клэй Колье (Clay Collier), занятый в создании программного обеспечения для автомобильных систем навигации, поведал историю о системе, над которой он работал по заказу крупной японской автоэлектронной компании. Клэй разработал по просьбе своего клиента прототип системы навигации. Как и подобает хорошему прототипу, он доказывал, что система будет работать, но в целом программа была едва функциональна. Однажды в США прилетел президент этой японской компании – производителя автомобильной электроники. Президент желал увидеть программу в действии. Коллега Клэя, назовем его Ральф, знал, что президенту из Японии отказать нельзя, придется сотворить демонстрацию. Ральф встретил президента в аэропорту на автомобиле, специально оборудованном прототипом навигационной системы. Ральф знал, что прототип может указать дорогу к офисам компании в Лос-Анджелесе, но остальные адреса даже не проверялись. К огорчению Ральфа, президент попросил отвезти его на ланч в конкретный ресторан. Ральф не знал, где находится ресторан, и вовсе не был уверен, что прототип сможет указать туда дорогу. Он скрестил пальцы и набрал название ресторана. К его удивлению, компьютер начал давать указания: «Повернуть направо на Линкольн-стрит», «Перестроиться в левый ряд», и так далее. Ральф послушно следовал указаниям, в то время как президент молча думал о чем-то своем, однако вскоре инструкции привели их в сомнительные районы города, так что Ральф забеспокоился. Его беспокойство достигло предела, когда он остановил машину по команде компьютера, и в этот момент кто-то распахнул дверь автомобиля снаружи. К бесконечному облегчению Ральфа, дверь открыл сотрудник ресторана. Президент расплылся в улыбке.

Успех демонстрации прототипа обернулся для Ральфа неприятностями. Президент настолько впечатлился работой системы, что захотел, чтобы Ральф превратил ее в готовый продукт. Ральф возразил, что прототип недостаточно надежен, чтобы стать основой миллионов устройств, но президент ничего не хотел слышать – он же видел, что прототип работает. Ральф согласился, и восемь долгих лет спустя его компания, наконец, поставила первую работающую версию продукта. Она работала медленно, с ошибками и уже не могла угнаться за новыми, более сильными конкурентами. Газета New York Times назвала его «очевидно слабым».

Компетенция и знания, приобретенные Ральфом и его командой в процессе создания *неправильного* прототипа, были гораздо более ценны, чем код. Президент этого не понял, оценив код выше, и в результате пострадала вся компания.

* * *

Если определять границы проекта разработки лишь в терминах фиксированных сроков сдачи и перечня функций, даже своевременная сдача продукта не сделает его желанным. Если же определять проект в терминах качества и удовлетворения потребностей пользователей, вы получите востребованный продукт, и сроки разработки не будут более длительными. Старая шутка Кремниевой Долины: «Как сделать небольшое состояние на программном обеспечении?» И ответ, конечно: «Начать с большого состояния!» Скрытые издержки проекта по разработке программного обеспечения, даже при опытном руководстве, достаточно велики, чтобы даже Дональд Трамп задумался. Гонки на яхтах и пристрастие к наркотикам в долгосрочной перспективе обходятся дешевле, чем неконтролируемое создание программного обеспечения.

Глава 4
Танцующий медведь

Даже если уцелевшие осознают, что именно интерактивный продукт заставляет их чувствовать себя глупо, они находятся в окружении апологетов и, как правило, не могут

говорить об этом, не выставляя при этом себя нытиками. Ворчунов не любят, поэтому люди испытывают сильное социальное давление, вынуждающее их присоединиться к поборникам, принести извинения, обвинить себя в плохой расторопности. Однако инстинкты уцелевших пользователей сильнее сознательных попыток компенсировать чувство неуверенности. Программы *заставляют* их чувствовать себя глупо, хотя так не должно быть. Если вы из таких людей, то, возможно, задаетесь вопросом: «Что он имеет в виду, говоря о некачественных программах? Ведь эти программы решают рабочие задачи?» В оставшейся части главы я опишу свое понимание качества.

Если это проблема, то почему ее до сих пор не решили?

Танцующие программы-медведи плохи тем, что большинство людей довольствуется неуклюжими танцами. Лишь увидев, как должен выглядеть настоящий танец, они начинаются подозревать, что за медвежьим шарканьем скрывается иной мир. Очень немногие из продуктов, основанных на программном обеспечении, демонстрируют настоящие танцы, и большинство людей даже не подозревает, что ситуацию можно улучшить, причем существенно. Большинство пользователей электронных таблиц и текстовых процессоров на современных компьютерах считают, что все проблемы, которые способен решить компьютер, *уже были решены,* и найденные решения как минимум адекватны. Такое представление далеко от истины. Бесконечное число задач, связанных с обработкой информации, еще не решено, а в большинстве случаев никто даже не рассматривал возможные решения.

Жертва бытовой электроники

Как потребители продуктов, основанных на программном обеспечении, мы настолько привыкли принимать как должное все, чем мы пользуемся, что не можем увидеть, что мы должны иметь. Инженеры создают продукты, выполняющие задачи, из которых состоит работа, однако без надлежащего проектирования этот набор задач не позволяет пользователю достичь своих целей.

За двадцать лет у меня было множество видеомагнитофонов. Каждый имел функцию записи передач в указанное время, и ни один, включая модель за полторы тысячи долларов, не давал полной уверенности, что у меня все получится. Интерфейс этого продукта настолько неудобный, настолько сложный в интерпретации, настолько расплывчатый в терминологии и настройках, настолько переполнен скрытыми режимами и переключателями, что мне удавалось осуществить запись лишь в четырех случаях из десяти. Более чем в половине случаев я обнаруживал, что записал три часа бразильского футбола вместо передачи с канала Пи-Би-Эс. Проведя в борьбе долгие годы, я признал поражение и больше даже не пытаюсь записывать телепрограммы. Как и все остальные члены моей семьи. Как и все мои друзья. Мы – люди, уцелевшие после столкновения с танцующими программами-медведями.

Пребывая в отчаянии, я отправляюсь в местный Электронный Рай с «визой» в кармане. «Тысяча! Нет, две! – кричу я. – Награда продавцу за видеомагнитофон, который я смогу использовать для записи телепередач!» Люди в сияющих костюмах собираются вокруг меня и предлагают свой товар. От низкобюджетных вариантов до самых дорогих аппаратов и нет никакой разницы во взаимодействии. Конечно, в каждом большой набор возможностей, но способ управления устройством одинаков независимо от цены. Иначе говоря, продукт совершенствуют уже двадцать лет, а пользоваться им мне ничуть не легче. Это и есть танцующие программы-медведи в своем лучшем виде.

Когда я говорю об этом продавцу, он, защищаясь, сообщает мне, что лучше все равно не бывает. Он показывает мне место в брошюре, где написано, что видеомагнитофон «прост в использовании». Билл Гейтс однажды заметил, с нетипичным для него цинизмом, как

сделать программу дружелюбной к пользователю: изготовить печать и поставить на каждой коробке штамп «USER FRIENDLY» (Дружелюбна к пользователю). Компьютерная индустрия *взяла* его метод на вооружение.

Кнопочное управление не очень справляется с такой непрерывной сущностью, как время, в отличие от вращающегося манипулятора. Будь у этого видеомагнитофона дисковый манипулятор, как у моего будильника за одиннадцать долларов, я смог бы устанавливать время и навсегда избавиться от мерцающих цифр «12:00». Второй такой манипулятор для указания времени записи следующей передачи и я бы уже с легкостью записывал то, что меня интересует. Реальность же такова, что устройства, обладая возможностями для программирования записи *десяти* передач, столь неудобна, что не позволяет записать и *одну* из них.

Мы окружены танцующими продуктами-медведями. Упаковочные коробки с ног до головы исписаны их функциями. У них достаточно возможностей, чтобы заполнить колонку таблицы журнального обозревателя словом «да». Но они не делают пользователей счастливыми и не приносят им радости эффективной работы. Большинство не способно справиться со всеми возможностями и функциями. Это удается лишь апологетам, с радостью меняющим собственные привычки, чтобы справиться с вызовом, который им бросает программное обеспечение. Они наслаждаются возможностью повозиться с программой. Они трудолюбиво изучают новые возможности, которыми никогда не воспользуются.

Чем плохи почтовые клиенты

Пока производители устраивают одну за другой решительные битвы на рынке программного обеспечения, пользователи дрожат по своим рабочим отсекам в страхе ступить на неисследованную территорию. Например, производители электронной почты добавляют в свои списки функции одну за другой, не замечая базовых потребностей участников электронного обмена информацией.

Новых пользователей электронной почты завораживает новообретенная способность общаться напрямую, без сложностей и асинхронно с любым другим человеком. Однако решение задач не всегда позволяет пользователю достичь своих целей, и поэтому обмен электронной почтой все еще не вылез из пеленок. Проблема заключается в недопонимании реального применения электронной почты. Двадцать лет назад появление *любого* электронного письма становилось важным событием. Поскольку способ передачи подчеркивал важность сообщения, само сообщение ничего особенного собой не представляло. Более того, это был отдельный простой файл, состоящий из символов набора ASCII, не имеющих специальных свойств и связей.

Сегодня каждый из нас получает смешанный поток важных и бесполезных писем. Любой пользователь электронной почты быстро выясняет, насколько это мощный и полезный транспорт, и начинает активно пользоваться этим средством, чтобы решать повседневные и деловые задачи. Многие пользователи электронной почты получают десятки и сотни сообщений ежедневно. Сообщения в большинстве случаев отправляются либо в ответ на уже имеющиеся, либо с целью получения ответа. Сообщения таких последовательностей, или нити, передаются в обе стороны, связывая двух или более человек. На моем компьютере отношение связанных сообщений к одиночным составляет примерно 50:1. И при этом ни одна существующая программа для обмена электронными сообщениями не считает сообщения электронной почты фрагментами последовательностей[10]. Они ведут себя так, словно нити не существуют или являются незначительным свойством редких сообщений.

[10] Некоторые программы дают пользователю возможность вручную создавать нити и управлять ими, однако лекарство оказывается хуже болезни. Этой возможностью непросто управлять, и нити общения по-прежнему считаются чем-то исключительным.

Легко понять, что просмотр нитей вместо сообщений позволит пользователю отчетливо прослеживать связи и потоки информации в сообщениях и то, как они формируют связное общение. Если рассматривать проблему с точки зрения функций, мы увидим лишь потребность в ответах на сообщения и отправке сообщений.

Работа с нитями электронных сообщений – не особо сложная задача с точки зрения программирования; все дело в том, что никто и никогда не создавал такие программы, а программисты с неохотой реализуют нововведения, исходящие от пользователей.

Рассматривая программное обеспечение с точки зрения реализации, программисты видят, что сообщения передаются между пользователями и что пользователи могут складывать сообщения в папки, так что программисты проблем не наблюдают. Когда медведь уже начал двигаться, они объявляют это танцем и прекращают дальнейшую работу.

Электронная почта – лишь один из примеров программного обеспечения, не позволяющего достичь простых и очевидных первоочередных целей. Нас так впечатляют танцующие медведи, что мы не видим неадекватности подобных продуктов. Вот еще несколько примеров.

Чем плохи программы для планирования

В офисе адвоката, в рекламном агентстве, в кабинете бухгалтера, в любом другом предприятии, связанном с консультированием, существует серьезная и не удовлетворяемая потребность в программе, управляющей распределением людей по проектам с учетом времени. Такова организация деятельности всех консультирующих компаний, и все же, как это ни поразительно, не существует программы, решающей эту задачу.

С точки зрения программиста управление проектом – это задача планирования плюс, возможно, дополнительные навороты вроде анализа методом критического пути, если начало одной задачи зависит от завершения предыдущей. Все доступные сегодня программы управления проектами основаны на этом академически чистом предположении[11]. Проблема в том, что этот взгляд на управление проектами имеет очень слабое отношение к действительности.

Ошибочно одно из фундаментальных предположений программ для управления проектами, а именно: людям необходимо помочь понять, как надо действовать в рамках проекта. Большинство людей довольно успешно справляются со своими проектами; ведь такова их работа, в конечном итоге. Увязывание нескольких проектов в единое расписание – вот в чем действительно требуется помощь. Ресурсы (обычно подразумеваются люди) одновременно задействованы в нескольких проектах. Они начинают и заканчивают проекты один за другим, иногда с некоторым наложением, так что большая часть проектов стоит в очереди, ожидая своей судьбы. Недостаточно распределить людей по проектам, необходимо иметь возможность назначить одного человека на работу в нескольких проектах.

Чтобы приносить пользу, такие программы управления ресурсами должны интегрировать три измерения проблемы: время, проекты, ресурсы. Вместо этого мы получаем программы, работающие лишь в двух измерениях: времени и ресурсов, причем продавцы программ настаивают, что больше нам ничего не нужно. Хоть под каким названием – «трафик», «управление проектами и „распределение ресурсов“, – этот жизненно необходимый сегмент рынка приложений попросту не существует.

Более того, проекты постоянно изменяются с учетом плана. Любая полезная программа для управления проектами должна быть гибкой и уметь адаптироваться к изменениям. Система управления проектами, не содержащая надежных и действенных механизмов обратной связи, позволяющих людям, выполняющим работу, указывать системе на истинное

[11] В этом смысле я не лучше остальных программистов. В 1984 году я написал SuperProjects для Computer Associates, одну из первых программ управления проектами. Как и практически все другие программы, появившиеся позже, она полностью игнорировала вопросы взаимодействия нескольких проектов.

положение вещей, не очень полезна в реальном управлении проектами.

Чем плохи календари

Практически каждый использует календарь для делового планирования. На рынке существует множество программ-календарей, и каждая из них игнорируют самый простой и очевидный способ применения календарной системы. Говоря просто, календарь должен отражать то, как люди используют время для управления своей жизнью. В целом, можно поделить наши заботы о времени на две категории: сроки и процессы. Срок – это момент времени, до которого что-то должно быть готово, скажем, промежуточная версия продукта. Пример процесса – деловая поездка на один день. Скажем, в ходе моего двухдневного визита в Чикаго у меня назначено три встречи с различными клиентами.

Все современные программы-календари игнорируют сроки и процессы, вместо этого пропагандируя встречи. Встреча – это отдельное событие, которое начинается в определенное время. Встречи – важная составляющая управления временем, но никоим образом не единственная. Речь не только об отсутствии других видов календарных записей, но и, во многих случаях, о неверном представлении собственно назначенных встреч.

Отслеживать начало встречи гораздо важнее, чем ее конец, однако календари не различают эти два момента времени. За последние тридцать лет я был инициатором и участником тысяч встреч. Время начала этих встреч имеет огромное значение. Время же завершения в большинстве случаев не важно, указывать его не требуется и оно не известно заранее. Однако в каждой из опробованных мной программ-календарей встречи обладают временем завершения, которое требуется указать заранее столь же точно, аккуратно и старательно, как время начала. Это время завершения используется в точных вычислениях вашей доступности, которые в принципе не могут быть точными, а потому являются серьезным искажением действительности. К примеру, если вы, пользуясь типичной календарной программой, пригласите меня на встречу в три часа дня, программа отвергнет ваше приглашение, если у меня в 2:30 назначена встреча, которая продлится 35 минут. В реальной жизни я могу с легкостью сбежать с предыдущей встречи на пять минут раньше.

Кроме того, ни одна из этих программ не учитывает время, которое мне необходимо, чтобы добраться до места встречи. Если мне нужно подъехать на другой конец города к двум часам дня, я должен выйти из дома в половине второго. На какое время я должен назначить встречу, на 1:30 или на 2? Качественно спроектированная программа должна решить этот вопрос самостоятельно и помочь мне выйти вовремя.

Есть и целый ряд других видов записей, связанных со временем, не учтенных в существующих календарях. В любой день у меня может быть десяток или более текущих проектов, однако в любой момент времени я работаю лишь над одним из них. Типичная календарная программа отказывается уживаться с этим моим нормальным поведением, не позволяя работать с календарем на уровне проектов. Я не могу обойти этого танцующего медведя.

Массовая веб-истерия

Любому владельцу компьютера и модема Всемирная паутина дала в руки удивительный ресурс. Среда Web – колоссальной силы инструмент, и он предлагает невероятные возможности. Удивительно, но самым важным достижением этой среды стала демонстрация того, насколько простым в применении может быть программное обеспечение. Многие бывшие апологеты находят Всемирную паутину столь простой в использовании, что требуют подобного свойства и от всех остальных программ. В частности, им нравится, что броузеры позволяют избегать раздражающего процесса установки.

Руководители индустрии ПО, особенно поставщики корпоративных приложений, с легкостью готовы запрыгнуть в этот вагон. Они влюблены по уши в программы, работающие

из броузера, поскольку могут распространять свои продукты, не мучая пользователей тошнотворным процессом установки. До эпохи Всемирной паутины все программные продукты требовали сложного процесса установки, тогда как продукты, работающие в рамках броузера, ничего такого не требуют. И большинству менеджеров индустрии программного обеспечения это кажется технологическим прорывом, превосходящим изобретение застежки-молнии.

Но это же просто притворство. Нет причин, по которым процесс установки не может быть невидимым для любой программы, независимо от технических деталей ее реализации. Если бы компьютер требовал обязательной установки программного обеспечения, то требовал бы независимо от наличия броузера. *Единственная причина по которой работающие вне броузеров программы требуют установки, состоит в том, что программисты привыкли всегда так делать в прошлом.* Их работа упрощалась, если они переносили ряд вопросов в программу установки. В ранних броузерах не было функциональности, позволяющей задавать такие вопросы, так что программисты просто пожимали плечами и переставали эти вопросы задавать. Если нужны еще доказательства: программисты практически не замечали этого недостатка, тогда как для многих пользователей отсутствие ее сделало Всемирную паутину самой простой в использовании платформой.

Если отвлечься от вопросов установки, браузеры слабы, как новорожденные котята – доисторические идиомы взаимодействия с человеком, архитектура, напоминающая плоскую шутку, гибкость не лучше, чем у сосулек. Любая программа, работающая внутри браузера, обязана принести в жертву невероятную производительность и возможности. Меня приводит в ярость желание некоторых руководителей вырезать своему приложению сердце, перенеся его на платформу Web, чтобы получить преимущества, связанные с отсутствием процесса установки, когда они могли бы получить то же самое, просто вежливо попросив своих разработчиков избавиться от этого процесса.

Пользователи требуют программное обеспечение, работающее в браузерах, потому что не знают лучших альтернатив. А вот разработчики программного обеспечения подчиняются этой тенденции, исходя из совершенно неверных предположений. Среда Web по организации схожа с Советским Союзом, здесь центральные компьютеры указывают беспомощным настольным машинам, что делать. Программисты – особенно в корпоративных ИТ-подразделениях – владеют центральными компьютерами, так что, подобно советским комиссарам, желают получить преимущества такого перехода. Вместо того, чтобы бесплатно получить преимущества программ, которые не надо устанавливать, пользователи расплачиваются потерей долговременного контроля над своей информационная инфраструктурой.

Что не так с программным обеспечением?

Большая часть первой моей книги посвящена подробному ответу на этот вопрос. Но я хотел бы занять еще несколько страниц, чтобы познакомить читателей с началами принципов проектирования взаимодействий, которые позволяют создавать более качественные программные продукты.

Программы забывают

Каждый раз, работая с программой, вы узнаете что-то о ней, но сама программа не запоминает ничего. Наш коллега, медиа-продюсер Трой Дэниэлс практически живет в Adobe Photoshop, который не помнит ровным счетом ничего из того, что он когда-либо делал. Не помнит, где Трой хранит файлы своих изображений, не помнит, как он обычно работает с этими файлами. Управляющие элементы и команды, которые он использует постоянно, в интерфейсе программы выглядят так же, как и остальные управляющие элементы и команды,

которые он не использует и вряд ли когда-нибудь начнет.

Программы ленивы

Прикладные программы, в большинстве своем, не сильно напрягаются для пользователей. Не в том смысле, что не делают работу, но в том смысле, что часто тратят гигантские усилия, чтобы ублажить пользователей, относясь к ним так, как если бы они были программистами. Это все равно, что подарить жене на день рождения электродрель. Если электродрели нравятся *вам,* это совсем не означает, что они нравятся и *ей.* Направив усилия программистов на создание того, что будет действительно нужно пользователю, мы сможем сделать пользователя гораздо более счастливым, не вынуждая разработчиков при этом трудиться больше.

Программы скупы на информацию

Подобно банкомату, скрывающему от меня количество денег на счете, большинство интерактивных продуктов скупы на информацию. Они склонны не только к сокрытию информации о происходящем, но и *маскировке собственно происходящего.* Типичный пользователь интерактивной системы не может определить ее состояние, пока она не выпалит сообщение, указывающее на окончательный сбой. Для примера можно взять мои новые радиочасы, описанные в главе 1. Загадки века информации, которые морочат мне голову, скрывая свое состояние. Внешне все выглядит так, будто система работает замечательно, но это не так, и нет способа узнать правду.

Если, работая с программой, вы заметите, что делаете заметки на полях страницы, оказавшейся поблизости, знайте, что стали жертвой программы, скупой на информацию. Любая программа могла бы с легкостью отображать намного больше полезной информации, но немногие программисты об этом задумываются. К примеру, когда моя программа электронной почты принимает новое сообщение, она отображает крохотную пиктограмму конверта. Этот маленький конверт может означать, что у меня одно новое сообщение, но может означать и тысячу новых сообщений. Программа не позволяет мне оценить глубину цифрового почтового ящика. Эта скупость препятствует оценке ситуации в целом.

Программы не гибки

Когда человек имеет возможность оценить ситуацию в целом, он часто адаптируется к этой ситуации, исходя из оценки. Программное обеспечение редко бывает настолько гибким. Когда клерк видит, что стопка бумаг для обработки достигла пятнадцати сантиметров в толщину, он понимает, что следует предпринять какие-то решительные меры, чтобы избежать завала. Программы в большинстве своем написаны так, что человек может видеть лишь бумагу наверху стопки, но не далее. Независимо от метафорической глубины папки входящих документов, компьютер ведет себя так, как если бы только одно сообщение требовало ответа. Обратное тоже верно. Если в реальных входящих только одна бумага, человек может воспользоваться паузой и помочь коллеге с более высокой стопкой. Компьютер на такое не способен.

При компьютеризации ручного труда программисты (или аналитики) изучают существующее поведение пользователей, выполняющих ручную работу, и делают выводы относительно задач и функций. Далее задачи реализуются в компьютерной программе. Как правило, все аспекты работы, не связанные с задачами, попросту теряются в процессе.

В традиционной системе с ручным трудом управляющий может вытащить заявление своего близкого родственника со дна стопки и ускорить обработку этого заявления. Как вариант, рассерженный позвонивший, который ведет себя грубо, может добиться того, что его заявление переложат в самый конец очереди. Такая системная гибкость – ключ к

сохранению социального порядка. Компьютеризованной системе присуща холодная рациональность, разрушающая структуры цивилизации.

Люди-пользователи предпочитают системы, допускающие легкое жульничество. Им нужна возможность слегка покачнуть пинбол-автомат, чтобы сдвинуть игру с мертвой точки, не сильно, но достаточно ощутимо, чтобы получить положительное влияние на исход игры. Именно эта податливость делает системы ручного труда столь эффективными, пусть и более медленными, в сравнении с компьютеризованными.

Программы возлагают вину на пользователей

Если программа сталкивается с проблемой, она неизменно возлагает проблему на пользователя, да ко всему еще и обвиняет его в возникновении проблемы. Если с человеком что-то случается, он, как правило, сознательно пытается компенсировать последствия, к примеру, если за обедом у приятеля я разолью чей-то бокал вина, то воспользуюсь своей салфеткой, чтобы остановить распространение пятна, а затем налью пострадавшему новый бокал. Я проявлю заботу и внимательность, никто не обидится, и ни у кого не останется плохих впечатлений от этого происшествия.

Когда интерактивный продукт сталкивается с небольшой проблемой, то часто впадет в бездействие, превращаясь в бесполезную инертную массу. Такой сбой часто приводит к многочисленным сопутствующим повреждениям. Скажем, программа установки задает пользователю ряд вопросов, прежде чем начать копирование программы на жесткий диск. В прежние времена, если дисковое пространство заканчивалось в процессе установки, это приводило к сбою программы установки. Современные программы не намного лучше. Если заканчивается место, они могут выдать сообщение об ошибке, но затем перестают работать, забывая обо всех настройках, которые вы столь тщательно сделали. Если, расчистив дисковое пространство, вы снова запустите установку, она в первую очередь задаст вам все уже заданные вопросы, поскольку не запоминает ответы.

Программы не несут ответственности

Диалоги подтверждения – один из наиболее распространенных примеров некачественного проектирования; они спрашивают, «уверены ли мы», что хотим выполнить то или иное действие. На заре компьютеризации рабочих мест программы выполняли необратимые действия в ту же секунду, когда пользователь вводил команду. Команда «erase all» (стереть все) делала именно это, причем немедленно и необратимо. Как только первый пользователь непреднамеренно стер все содержимое жесткого диска, он, несомненно, пожаловался программисту, и программист добавил адекватный, с его точки зрения, уровень защиты. Пользователь набирает команду «erase all», и теперь, прежде чем компьютер ее выполнит, программа просит пользователя подтвердить команду на удаление.

Все очень логично и совершенно неправильно.

Диалог подтверждения – удобный выход для программиста, поскольку избавляет его от ответственности за содействие непреднамеренному удалению. Однако здесь имеется неправильное понимание источника проблемы. Удаление целиком лежит на совести пользователя, и он уже набрал команду. Теперь не время программе проявлять нерешительность. Она должна просто взять и выполнить возложенную на нее задачу. В действительности имеет место уход от другой ответственности – ответственности программы быть готовой *отменить* действия, пусть даже пользователь захотел их выполнить.

Люди обычно принимают решения иными способами, нежели компьютеры, так что для человека нормально и типично передумать или захотеть отменить принятое ранее решение. В реальном мире за пределами компьютеров большинство действий можно отложить, изменить, обратить. Не существует причин, по которым такое поведение не может быть

реализовано и в продуктах, основанных на программном обеспечении; просто создающие их программисты об этом не задумываются.

Банкомат из главы 1 отказывается от ответственности при помощи подтверждений точно так же, как программы для настольных компьютеров. Когда я вставляю карту, банкомат требует подтвердить, что я вставил карту. Когда я запрашиваю наличные средства, он требует подтвердить, что я хочу снять деньги. Когда я набираю сумму, требует подтвердить, что я набрал сумму. Почему бы машине просто не довериться мне? Почему просто не выполнить транзакцию?

Она может дать мне возможность прервать транзакцию в любой момент гораздо более простым способом. Будь у этого банкомата обычная большая красная кнопка ОТМЕНА, которую я мог бы нажать в любое время, банкомат мог бы предполагать, что я разумен, осознаю, чего хочу и что делаю, вместо того чтобы считать меня глупым, некомпетентным и не имеющим четкого представления о своих желаниях.

Уверен, что некоторые из пользователей этого банкомата действительно глупы и некомпетентны, однако никому – даже глупым и некомпетентным людям – не нравится, когда их считают глупыми и некомпетентными. Кроме того, подобное отношение к клиенту никогда не вызывает у него привязанности и положительных эмоций.

Исправить эту проблему несложно. Программа должна поместить слова «Снятие со счета» вверху экрана и *не стирать их на протяжении всей транзакции.* Затем она должна отобразить на экране «$1.50» – стоимость операции – *и также не стирать.* Затем она должна добавить слова «Идет проверка» наряду с номером моего счета, балансом и ограничениями на изъятие средств и *оставить всю эту информацию на виду.* Таким образом, когда зайдет речь о сумме, я буду уже хорошо информированным потребителем, а не жертвой допроса. Я могу принять решение: указать сумму, обладая знанием об ограничениях, о средствах, о возможностях и о приличиях.

Система, представляющая полезную информацию, подобная описанной только что, суть типичный пример того, как работают человеческие системы, потому что людям необходимо видеть картину в целом. С другой стороны, компьютеры нуждаются лишь в небольших фрагментах информации, чтобы переходить от одного этапа процесса к другому. И именно на этой основе моделируется взаимодействие: система предполагает, что пользователь, нажимающий на кнопки, пока его друзья нетерпеливо переминаются с ноги на ногу, – лишь еще один компьютер, а не теплокровный человек, способный чувствовать.

* * *

Многие новички в мире компьютеров воображают, что программное обеспечение ведет себя так, как ведет, потому что на то есть уважительная причина. Напротив, поведение программ часто есть результат прихотей или случайностей, которые бездумно повторяются из года в год. Добавив в процесс разработки продуктов, основанных на программном обеспечении, своевременное проектирование взаимодействия, мы сможем получить значительные преимущества.

Глава 5
Нелояльность клиентов

Истинная выгода от хорошо спроектированного продукта или услуги – бесконечная преданность, которую такой продукт пробуждает в ваших клиентах. В этой главе я покажу, как эта преданность может послужить для компании спасательным кругом в тяжелые для бизнеса времена и снабдить ее оружием против конкурентов. Я также покажу, насколько вы уязвимы без этой преданности.

Привлекательность

Ларри Кили (Larry Keeley) из Doblin Group создал занимательную концептуальную модель трех важнейших качеств в бизнесе высоких технологий. Кили называет первое качество потенциалом – его источником служат инженеры. Они спрашивают: «На что мы способны? Что возможно?» Инженеры должны знать, что можно создать и чего создать нельзя. Продукт не может стать успешным, если его невозможно создать и заставить работать.

Второе качество Кили называет жизнеспособностью – это вклад деловых людей. Они спрашивают: «Что окажется жизнеспособным? Что мы сможем продать?» Деловые люди должны знать, что можно и чего нельзя создать и прибыльно продать. Продукт не может стать успешным, если не может поддержать растущую компанию.

Поскольку всем успешным высокотехнологичным предприятиям требуется баланс этих двух качеств, связь между ними очень крепкая. Бизнесмены полностью зависят от способности инженеров создавать работающие вещи. А инженеры полностью зависят от способности бизнесменов их обеспечивать соответствующими инструментами. Это сложный союз.

Программисты любят добавлять функции в свои продукты. Заставить ядро программы работать с максимальной эффективностью – для них творческий вызов. Это проявление потенциала, и некоторые инженеры счастливы, даже если конечный продукт никогда не появится на рынке. Когда нанявшая их компания терпит неудачу, они просто меняют место работы. Их личный успех не зависит от успеха предприятия.

Бизнесмены же любят захватывать доли рынка и продавать разнообразные товары. Их сверхзадача – пробудить в людях тягу покупать эти продукты. Это проявление жизнеспособности, и некоторые деловые люди могут быть счастливы, даже если продукт не слишком сложен технически. Большинство деловых людей вполне удовлетворится продажей камней-любимцев, если те будут продаваться в достаточных количествах.

Эти две стороны зависят одна от другой, но их разнящиеся цели создают структурный изъян отношений. Симбиоз столь же нестабилен, как табурет на двух ножках. И здесь на сцене появляется третье качество, предложенное Кили, которое играет роль третьей ножки табурета.

Третье качество Кили называет привлекательностью, и за эту составляющую отвечают проектировщики. Они должны спрашивать: «Что привлечет людей? Чего они хотят?» Проектировщики должны знать, что сделает людей счастливыми и даст им удовлетворение. Продукт не может иметь долгосрочного успеха, если не дает удовлетворения и ощущения могущества людям, для которых он предназначен.

Проектирование делает продукт, который а) может быть создан и может хорошо работать и б) может распространяться и прибыльно продаваться, продуктом успешным, путем в) превращения его в то, чего *действительно хотят люди.* Эта третья ножка служит стабилизатором и преобразует интересные технологические достижения в долгосрочный успех.

И хотя существует возможность выделить привлекательные качества существующего продукта, Кили считает, и я с этим согласен, что разумнее *сначала* решить, что находят привлекательным покупатели, а затем поставить перед инженерами и деловыми людьми задачу создания и продажи такого продукта. Этот подход может дать огромные преимущества смышленому игроку. Он позволяет вам оторваться от конкурентов. Пока каждый из них в этой стае реагирует на конкурирующие ходы соперников, борется с вопросами «возможно ли?» и «жизнеспособно ли?», вы вырываетесь вперед, сосредотачиваясь на еще неудовлетворенных потребностях своих покупателей. Пусть ваши конкуренты сражаются между собой, в то время как вы занимаетесь обеспечением самых насущных потребностей клиентов.

Для примера: в начале девяностых годов одним из серьезных игроков на рынке программ для Windows была компания Borland International. У меня была возможность изучить ее бизнес, когда я занимался в этой компании консультированием. В компании существовал замечательный альянс высокопрофессиональных деловых людей и очень сильных разработчиков программного обеспечения. Едва ли не каждый день я знакомился с еще одним впечатляющим экспериментальным проектом. Во главе каждого проекта стояла пара – бизнесмен экстракласса и не менее одаренный разработчик.

Проекты были похожи: классная технология, очевидная рыночная ниша, очевидный коммерческий потенциал, блестящие умы. Впечатление от такого количества талантливых людей и замечательных проектов поначалу было поразительное. Но через какое-то время стала очевидной истинная природа этих проектов: лишь очень немногие из них завершились выпуском продукта. Более того, мало в каких проектах были определены сроки сдачи. Низкие прибыли и высокие затраты, и после пяти лет перетягивания каната между потенциалом и жизнеспособностью для компании Borland вполне предсказуемо начались тяжелые времена, и она была вынуждена уволить большинство сотрудников.

В Borland, как и в большинстве современных высокотехнологических компаний, не было сколько-нибудь заметных талантов в области проектирования, равно как и существенного понимания роли проектирования в деловой и технической культуре. Поэтому Borland с трудом могла преобразовать свои жизнеспособные, обладающие серьезным потенциалом проекты в привлекательные продукты.

Привлекательность легко перепутать с потребностью, но это кардинально различные свойства. Меня *привлекает* возможность провести полтора месяца на Бермудских островах, но мне это не *нужно.* Если у меня желчные камни, то я *нуждаюсь* в операционном лечении, но меня это *не привлекает.* Риэлтор Салли *нуждается* в том, чтобы продать четыре дома в этом году. А возможность обеспечить четыре семьи счастьем и уютом ее *привлекает.* Она *нуждается* в MLS-программе, позволяющей опубликовать предложения своих услуг в многочисленных каталогах, но если эта программа будет заставлять ее чувствовать себя глупой, то она ее *не привлечет.*

На короткий срок человек может оказываться под мощным влиянием своих потребностей, но в долгосрочной перспективе его желания могут оказывать более сильное и более выраженное воздействие. Желания людей всегда находят выход после того, как удовлетворены потребности. Когда человеку что-то нужно, он сделает то, что необходимо, чтобы это получить, но если человека что-то привлекает, он *предан этому.* Он знает, что тратит свободные средства, оставшиеся после удовлетворения потребностей, и поэтому купит то, что приносит счастье, причем, не обязательно исходя из рациональных суждений. Когда потребитель желает владеть продуктом определенной торговой марки, его преданность сильнейшим образом влияет на бизнес.

Модель Кили показывает, как добиться преданности клиентов. Компания, разрабатывающая программное обеспечение, может быть *жизнеспособной,* лишь удовлетворив потребности риэлтора Салли. Но мы видим, что компания может стать сильнее, просуществовать дольше, даже возглавить индустрию, если *привлечет* Салли. Если

продукт просто соответствует потребностям Салли, она вынуждена стать либо апологетом, либо уцелевшей. В любом случае, несмотря на необходимость изучать программу, она не будет довольна программой и не станет рекомендовать ее коллегам. Но если продукт удовлетворяет *желания* Салли, он становится ее другом и помощником в ежедневной работе. Салли привязывается к программе, рассказывает о ней коллегам и друзьям. Она счастлива на работе и испытывает гордость. Так MLS-программа, повышая производительность труда и эмоциональный настрой Салли, пробуждает в ней лояльность к продуктам компании.

Продукт, спроектированный без учета привлекательности, может удовлетворить потребность рынка, но любой его успех станет успехом танцующего медведя. Самая большая слабость таких программ в том, что они никогда не пробуждают в клиентах лояльность. Компания, лишенная преданности покупателей, крайне уязвима для конкурентов.

Одно сравнение

Существует три известных высокотехнологичных компании, иллюстрирующих динамику треножной модели Кили, каждая со своими плюсами и минусами: Novell, Microsoft, Apple.

Недостаточная лояльность клиентуры в долгосрочной перспективе, как правило, ставит компанию на колени, несмотря на серьезность удовлетворяемых компанией потребностей рынка. Компания Novell – отличная иллюстрация этого тезиса. В начале девяностых годов единственным практичным способом объединения офисных компьютеров в сеть была система Novell NetWare. NetWare – как продукт – прошла испытание на потенциал, а Novell – как компания – прошла испытание на жизнеспособность. Потребность в локальных сетях в то время была просто невероятная, и никакой другой поставщик решений не смог ее удовлетворить. Некоторые компании, такие как Banyan и Corvus, также имели технические решения в этой области – и тоже прошли испытание на потенциал, однако провалили испытание на жизнеспособность – их бизнес-подход провалился. Ни одной из этих компаний не удалось создать привлекательный продукт, поэтому, несмотря на процветание Novell, лишь жажда немедленного удовлетворения потребностей побуждала клиентов создавать сети на основе NetWare, и эта система осталась нелюбимым танцующим медведем.

Компания Novell выросла толстой и самодовольной, но система NetWare была очень плохо спроектирована, поэтому установка и сопровождение требовали участия дорогостоящего специалиста. Более того, сети на основе NetWare вели себя грубо и неподобающе, приводя пользователей в отчаяние. Novell не смогла это осознать, вероятно, потому, что миллионы покупали NetWare, однако рост количества покупателей стимулировала именно потребность, а не привлекательность NetWare.

В начале девяностых годов компании Microsoft, 3Com и даже Apple начали поставлять на рынок продукты, обладающие тем же потенциалом, что NetWare, но не ставящие клиентов в жесткую зависимость от приходящих специалистов по установке и особенно по сопровождению. Novell в немом ужасе наблюдала, как испаряется ее лидирующее положение. Как только возникла конкуренция, сказалось отсутствие лояльности клиентуры компании Novell. Сегодня бизнес Novell[12] в основном заключается в поддержке тех клиентов, которые оказались технологически привязаны к компании. Новые клиенты выбрали другие компании.

Компания Novell была жизнеспособной и обладала крайне высоким потенциалом. Она владела мощной технологией и адекватной деловой хваткой, но пострадала из-за полного отсутствия проектирования взаимодействия.

[12] Впрочем, в индустрии высоких технологий положение дел может измениться очень быстро. Я пишу эти строки весной 1998 года, в то время как Эрик Шмидт, новый президент Novell, начинает реанимировать компанию.

* * *

История Microsoft проста. Продукты компании технически полноценны, но редко когда оказываются передовыми. Microsoft предпочитает стоять на плечах гигантов, уже проведших исследования и разработку[13]. Однако не вызывает сомнения, что Билл Гейтс является одним из самых талантливых бизнесменов своего поколения, а то и века. Он обладает удивительной способностью извлекать успех из практически каждого предприятия, несмотря на возникающие препятствия.

Microsoft не занимается или почти не занимается проектированием, а ее продукты известны тем, что заставляют пользователей чувствовать себя глупо. Они известны также большим количеством функций.

Многие деловые предприятия и профессионалы испытывают приверженность к продукции Microsoft, но в большинстве своем из-за экономической необходимости и вследствие отсутствия альтернатив. Немногие другие компании способны предоставить столь же полное решение, как Microsoft. Однако не следует путать экономическую необходимость с лояльностью клиентов. Очень немногие пользователи преданы Microsoft.

Microsoft – компания с некоторым потенциалом и удивительной жизнеспособностью. Microsoft обладает адекватной технологией и великолепным бизнесом, что в краткосрочной перспективе компенсирует отсутствие проектирования.

* * *

Лояльность клиентуры может стать для компании приобретением бесценным, и корпорация Apple по справедливости знаменита своим пристрастием к проектированию и дизайну на всех уровнях. Каждый аспект корпоративной индивидуальности Apple, ее продуктов и маркетинговых кампаний пронизан замечательным чувством дизайна. Награды и почести, воздаваемые Apple, слишком многочисленны, чтобы их пересчитать, но достаточно взглянуть на ее программы, устройства, упаковки и документацию, и даже просто на вечеринки, которые компания закатывает на выставке Microsoft World, и становится понятно, что проектирование и дизайн близки ей по духу.

Такая приверженность дизайну и внимание к деталям взаимодействия создали для Apple граничащую с фанатизмом – и часто переходящую в него – лояльность клиентов. Пользователи компьютеров Macintosh – самые преданные из всех владельцев продуктов, основанных на программном обеспечении. Никакой другой продукт или производитель не вдохновляет на такую личную преданность, как Apple. Клиенты компании наклеивают логотипы Apple на автомобили, заказывают специальные номерные знаки для своих автомобилей, носят футболки Apple, показывают свою принадлежность духу Apple. Они превозносят достоинства Маков любому, кто готов выслушать. В большинстве ситуаций компьютер Wintel удовлетворит все потребности человека лучше, дешевле и быстрее, чем Мак, но Macintosh всегда остается избранным компьютером. На недавней конференции по дизайну только одна докладчица использовала машину Wintel вместо Макинтоша, при этом она очень сильно извинялась перед аудиторией за свое «предательство» – ведь она предала компьютер, единственно достойный обладателя хотя бы зачатков чувства гармонии.

Технологическая доблесть Apple достойна, но лишена величия. С точки зрения технического потенциала Apple в плане инноваций не лучше Microsoft.

У Apple ушло двенадцать лет на потерю лидерства, которое Novell потеряла всего за год. Мало какие проблемы Apple были связаны с внешними факторами. Напротив, эта компания пострадала от поразительного разнообразия ею же самой созданных проблем. К примеру, в середине восьмидесятых Стива Джобса, основателя и главного энтузиаста

[13] В старой шутке говорится, что исследовательское подразделение Microsoft находится в Купертино. Это город в Кремниевой Долине, где расположен центр передовых технологий компании Apple.

компании, изгнали и заменили не имеющим отношения к компьютерам и употребляющим только безалкогольные напитки руководителем, который раз за разом принимал негодные с точки зрения бизнеса решения. Завышенные цены на продукцию, плохой маркетинг, злобно-пренебрежительное отношение к сторонним разработчикам программного обеспечения, и помимо прочего, Мак оставался закрытой платформой; последняя стратегия широко упоминается, как причина свержения других платформ, властвовавших на рынке (таких как VMS, MVS и OS/2).

Все эти грубые промахи легко уничтожили бы обычную компанию, но проектирование и дизайн, сделавшие Макинтош привлекательным, заработали Apple невиданную лояльность покупателей. Маки удовлетворяли потребности пользователей не лучше, чем компьютеры на платформе Windows, а во многих случаях даже хуже, но удовлетворение потребностей не является жизненно важной составляющей успеха на рынке.

После продолжительной критики в адрес руководства, явных финансовых потерь, после создания средненьких продуктов, после всех миллиардов, выброшенных на пустые исследования, после потери двух третей рынка ни одна другая компания по-прежнему не имеет более преданной армии клиентов. Это дает корпорации Apple огромные преимущества в ведении бизнеса. Многие из этих преимуществ сложно измерить, и ни одно из них не отражается в финансовых балансах компании, однако они столь же ощутимы и ценны для держателей акций, как чек на дивиденды.

Основанная на дизайне преданность заставляет поклонников Маков закрывать глаза на многочисленные преимущества продукции других производителей. Нежелание фанатов использовать другие продукты дает Apple время среагировать на инновации конкурентов. Лояльность покупателей дает Apple способность выдерживать сюрпризы, которые преподносят технологические прорывы. Падение Novell началось в тот же момент, когда конкурент – речь о Microsoft, – предложил жизнеспособный сетевой продукт. Гигантская доля рынка Novell ничего не смогла противопоставить натиску рыночных сил. С другой стороны, Apple, никогда не владевшая более чем 15% рынка компьютеров, упрямо сопротивляется натиску многочисленных конкурирующих компьютеров, мощных и дешевых.

Apple – компания, продукты которой привлекают пользователей. Приверженность Apple дизайну позволила компании пережить среднее качество технологии и губительное ведение бизнеса.

Учти Novell проектирование, компания могла бы преодолеть последствия слабых маркетинговых ходов. Если Microsoft когда-либо очнется и осознает значимость проектирования взаимодействия, конкурентам останется только сложить оружие и разойтись по домам. Apple в саморазрушении уподобилась звезде гранж-рока, однако, продолжая восстанавливать прежнюю форму, она может вновь стать жизнеспособной компанией.

Студентам, изучающим деловое администрирование в Гарварде и Стэнфорде, при изучении конкретных примеров ведения бизнеса редко указывают на значимость проектирования взаимодействия. Такое проектирование – обязательное условие успеха продуктов не только информационной, но и индустриальной эры, пусть там оно и проще. Кроме того, продукты индустриальной эры старше, присущие им проблемы и решения этих проблем уже давно изучены. В информационную эру – эру быстрых инноваций и крайне высокого когнитивного сопротивления – проектирование взаимодействия становится первоочередной необходимостью.

Время выхода на рынок

Когда та или иная компания захватила рыночную нишу, первой предложив востребованную функциональность, нет особого смысла торопиться выйти на рынок с эквивалентным продуктом. Вы уже проиграли гонку во времени выхода на рынок, и никакая скорость этого уже не изменит. Однако существует реальная возможность отнять лидерство

у первопроходца – при помощи более качественного проектирования. Проектирование делает ваш продукт желанным, и потенциальные клиенты будут активно стремиться к обладанию именно желанным продуктом, а не продуктом конкурента, независимо от того, кто первым вышел на рынок.

Компания, вышедшая на рынок первой, вынуждена принести несколько жертв. Велик шанс, что именно проектированием взаимодействия компания пожертвует[14]. Так что компания становится весьма уязвимой с точки зрения дизайна продукта, несмотря на быстрый захват рынка.

Быть же первым, кто добавил *новые* возможности, совсем не то же самое. Возможности не так выгодны пользователям, как изначальные механизмы решения конкретных проблем, поэтому добавление возможностей не даст столь же выгодного эффекта. На рынке, заполненном некачественно спроектированными продуктами, дополнительные возможности не помогут отвоевать заметного пространства[15].

Многие рыночные ниши заселены многочисленными производителями, продающими схожие продукты, ни один из которых не подвергался проектированию взаимодействия. Эти продукты состязаются, сравнивая функции. Всякий раз, когда один разработчик объявляет о новой возможности, все прочие добавляют эту возможность в следующие версии своего продукта. Эти ниши характерно балканизированы, раздроблены на многочисленные мелкие сегменты. Здесь нет доминирующего продукта или производителя. К примеру, на рынке персональных органайзеров присутствуют более десятка производителей. То же верно и для рынка сотовых телефонов.

Сражение между техническим потенциалом и жизнеспособностью может продолжаться многие годы, не давая пользователям расслабиться. Единственная сила, способная преобразовать раздробленный рынок, густонаселенный функциями и возможностями, в более стабильный, подчиненный проектированию рынок, – это сила внешняя. Такой внешней силой может стать по-бробдингнегски деловая хватка Билла Гейтса, а может, и грамотное применение проектирования.

Однако тяжкий труд Билла Гейтса по-прежнему не может пробудить любовь к его продуктам. Более того, средний уровень желанности практически всех высокотехнологических продуктов остается примерно на уровне продуктов Microsoft, несмотря на разум, искренность и тяжелый труд создателей. В следующем разделе я покажу, что за размножение неприятных и нежеланных танцующих программ-медведей ответственны простые, однако практически повсеместно существующие изъяны процесса создания продуктов, основанных на программном обеспечении.

Часть III
Как есть суп вилкой

Глава 6
Психбольница в руках пациентов

[14] Несмотря на свою приверженность проектированию, я поддерживаю такую тактику. Обнаружив совершенно пустую рыночную нишу, я бы безжалостно и с максимальной скоростью старался ввести свой продукт в игру. Однако сразу же после выпуска первой версии я бы сосредоточил все усилия на создании очень хорошо спроектированной второй версии. Если этого не сделаю я, могу поспорить, что это сделают конкуренты.

[15] Как указывает Джеффри Мур в своей великолепной книге «Crossing the Chasm» (Пересекая бездну), дополнительные возможности привлекательны лишь для первых пользователей, но не для рынка в целом.

Несмотря на различия типов продуктов, описанных в главе 1, все они обладают общим свойством – они раздражают. В этой главе я покажу, что источником такого положения дел является непреднамеренный захват власти в отрасли техническими специалистами. Если оставить в стороне маркетинговую риторику, форму продуктов для нас определяют люди, наименее для этой задачи подходящие.

Вождение на заднем сиденье

Показательна недавно опубликованная статья[16], посвященная впечатляющему провалу высокотехнологической начинающей компании General Magic. Автор затрагивает глубинную причину провала продукта, когда упоминает, что Марк Порат (Marc Porat), президент компании, «бросил все силы своей инженерной команды на проектирование устройства их мечты». Мишель Куин пишет без всякой иронии. Кажется совершенно естественным, что проектированием занимается команда инженеров, однако именно это и есть причина проблем. Позже в статье она цитирует одного из инженеров:

> «Мы так и не поняли, над чем работаем. Спецификация появилась лишь за восемь-двенадцать недель до завершения проекта».

И снова ни инженер, ни автор не замечают иронии. По общему тону статьи можно подумать, будто все сложилось бы лучше для General Magic, напиши инженеры черновики спецификации месяцем раньше.

Неважно, как рано в процессе разработки появляются спецификации, потому что они неспособны заменить проектирование взаимодействий, и неважно, насколько сильно программисты стараются, потому что они неспособны добиться успеха в проектировании взаимодействия. Кроме того, что их методы, опыт и способности не подходят для этой задачи, они еще и оказываются в центре конфликта интересов пользователя с трудоемкостью программирования. И тем не менее, раз за разом компании разрешают разработчикам программного обеспечения управлять процессом разработки, часто от начала и до конца проекта. Иногда этот контроль очевиден, но чаще осуществляется косвенно.

Я был свидетелем такого тонкого контроля в одной успешной, среднего размера компании Кремниевой Долины. На собрании присутствовал президент, весьма сведущий деловой человек, основавший компанию, а также ведущий программист, ответственный за создание продукта. Президент показал нам продукт и продемонстрировал его мощь, которая была для нас очевидна, как и то, что этой мощью сложно управлять – интерфейс продукта был чрезмерно сложен. Наша команда проектировщиков быстро поняла, что программисты «проектировали» этот продукт по ходу написания кода, – примерно так бобер «проектирует» свою плотину во время ее строительства.

Президент пожаловался, что конкурент с более слабым продуктом завоевывает рынок компании, но затруднился объяснить, почему это происходит, поскольку знал, что его собственный продукт мощнее. Президент пригласил нас, рассчитывая на нашу помощь в борьбе с конкурентом, но при этом наделил ведущего разработчика полномочиями делать то, что он сочтет уместным. Нам было ясно, что назрела отчаянная необходимость некоторой переделки поведения продукта, и мы рассказали, как мы себе это представляем. Для нас это была обычная и несложная работа по перепроектированию, в результате которой продукт этой компании за несколько месяцев стал бы гораздо более удобным и практичным, более мощным и приятным – более конкурентоспособным. Ведущий разработчик потряс нас просьбой *не вносить изменения во взаимодействия продукта с пользователем.* Он считал, что в этой области проблем нет. Ему казалось, что в положении продукта на рынке виноваты

[16] Мишель Куин (Michelle Quinn) «Vanishing Act» (Волшебство исчезновения), журнал *San Jose Mercury West* от 15 марта 1998 года.

недостаточно сведущие в его применении маркетологи компании. Он хотел, чтобы мы подготовили внутренние рекламные материалы, позволяющие маркетологам работать эффективнее. Он полностью отрицал наличие недостатков в продукте, несмотря их на неопровержимые свидетельства – в виде наступающего «более слабого» конкурента.

Программисты затрачивают столь много времени и энергии на изучение программного обеспечения, что для инженера казалось непостижимым, как пользователи могут не желать тратить время на изучение плодов его труда. Он с готовностью принимал версию, что источником проблемы является его компания, но полностью отрицал свою роль в создании этой проблемы. Он винил продавцов за то, что они не помогают покупателям изучить продукт. Он был готов работать, чтобы решить проблему, скажем, путем создания новых обучающих материалов, однако совершенно не считал возможными даже намеки на его собственное участие в сложившемся положении продукта.

Самодовольство инженера было поразительным. Гордость за создание такого мощного продукта ослепила его, но хуже того, ослепила и президента, который не видел неспособность инженера спроектировать продукт таким образом, чтобы пользователи остались довольны.

Продукт данной компании открыл новую нишу на рынке, внедрив новые методы сопровождения систем производства. Компания была быстрорастущей любимицей Уолл-Стрита и весьма удачно выпустила на рынок свои акции пару лет назад. Ее превозносили в деловой прессе и осыпали наградами общественные и коммерческие организации. Казалось, компания делает все правильно, и ее рыночная капитализация лишний раз это подчеркивала.

Но конкуренты наблюдают за подобным успехом не менее пристально, чем инвесторы, партнеры и сочувствующие. Конкуренты компании отчетливо видели потенциал рынка, и не менее отчетливо – слабость продукта данной компании. Они видели, насколько продукт мощный, насколько он насыщен возможностями, но видели также, что это просто танцующий медведь. Продукт имел передовую функциональность, но не мог осчастливить пользователей. Медведь танцевал, но танцевал плохо. Не нужно быть семи пядей во лбу, чтобы увидеть уязвимое место продукта, поэтому конкуренты просто скопировали многие из функций продукта, но сделали свой продукт более простым в применении. Отчеты, генерируемые этим новым продуктом, были прозрачны для руководителей и отражали динамику, тогда как отчеты в продукте-первопроходце были невразумительны и статичны. Конкурент-выскочка отобрал *шестьдесят процентов* рынка у первой компании – и это с менее мощным продуктом!

Наличие инженерных навыков помешало президенту компании. Упростив создание продукта, этот опыт встал на его пути, мешая увидеть заблуждения ведущего программиста. Глубоко укоренившись в программистской среде, он считал подобное положение вещей совершенно нормальным, тогда как наша команда была в изумлении. Этот президент не имел реальной власти. Его ведущий программист управлял делами компании подобно серому кардиналу.

Подготовка катастрофы

Мой коллега Скотт Мак-Грегор (Scott McGregor) поделился изложенной ниже историей, когда я спросил, знает ли он о случаях, когда проекты по разработке выходили из-под контроля из-за отсутствия проектирования взаимодействия. Его история печальна, и особенно тем, что типична для нашей отрасли.

Скотт – человек весьма талантливый, как видно из его хорошо написанной истории. Кроме того, он умелый проектировщик, с отличной родословной – академической и практической – как в разработке программного обеспечения, так и дизайне. Он присоединился к начинающей, с венчурным финансированием, компании в Кремниевой Долине. Основатели компании также имели достойное прошлое, включая несколько лет

успешной работы в Apple. Однажды Скотт пригласил меня познакомиться с основателями и рассказать им о моей компании. Президент компании и вице-президент по техническим вопросам показали нашей команде, над чем работает компания, и произвели на нас впечатление. Идея продукта была великолепной. Она основывалась на нестандартном взгляде на производственные процессы. Продукт включал в себя небольшое количество хороших технологий, которые позволяли удовлетворить вполне очевидную потребность рынка. У компании было все для успеха – но отсутствовала практика проектирования взаимодействия. Вот история, рассказанная Скоттом:

> Президент заявил, что мы побьем соперников, потому что действуем быстро и энергично. Затем с чувством собственного достоинства посоветовал нам следовать стратегии: «нечего тут думать – трясти надо», чтобы добиться успеха раньше всех. Разумеется, как только мы начнем трясти, нам на голову свалится что-нибудь тяжелое!
>
> Чтобы успеть сдать версию 1.2 к 31 декабря, мы просто приняли решение назначить версию 1.2 тому, что будет готово в пять вечера 31 декабря. Разработчики трудились, не имея фиксированной спецификации. Необходимость в значительных изменениях «без объявления войны» обнаружилась 29 декабря.
>
> Ранее я выдвигал мысль, что хорошо бы следовать какому-то методу проектирования. Я говорил, что сначала надо определить основные категории пользователей и заинтересованных лиц, создать для них профили, проработать определения их целей и задач, которые эти люди решают для достижения целей. Исходя из этих задач, мы сможем определиться с визуальным представлением ключевых объектов и способами взаимодействия с пользователями. И уже после этого сможем начать создание продукта.
>
> К сожалению, руководство единогласно посчитало такой подход непозволительной роскошью. Вместо этого мы ездили в гости к потенциальным клиентам, где наш президент делился планами на светлое будущее. Людям очень нравились идеи, и их интересовали конкретные детали. При этом каждый потенциальный клиент преследовал собственные цели, говоря о деталях. Одному нужен был продукт для отдела продаж, другому – для независимых реселлеров, третьему – для клиентов. Один пытался совладать с многочисленными документами, второму нужны были веб-страницы и т.д. Знакомство с каждым новым потенциальным клиентом увеличивало определение версии 1.2, превращая ее в перечень всех возможных функций.
>
> Что еще более прискорбно, потенциальные клиенты с удовольствием рассказывали о новых возможностях, которые хотели бы получить, но не о функциях, уже существующих в их программах или браузерах (то есть не о тех, что они уже воспринимали, как должное). Возможности, о которых не шла речь, не попали в спецификацию продукта, а потому так и не были реализованы.
>
> Наши только что принятые на работу вице-президенты по продажам и маркетингу неделями не могли установить продукт на свои компьютеры. Когда продукт, наконец, заработал, он уничтожал данные по нескольку раз на дню. Производительность продукта продолжала падать. В демонстрации с сотней записей производительность была низкой, но приемлемой, и разработчики не загружали систему сверх этого. Но реальные условия применения требовали тысяч записей, и в этом случае система работала со скоростью улитки.
>
> В продукте было три основных экрана, но чтобы просто отредактировать документ, необходимо было несколько раз переключаться между ними. Многие простые задачи требовали, чтобы пользователь раз десять щелкнул мышью, открывал и закрывал окна, постоянно переключаясь между мышью и клавиатурой.
>
> В конечном итоге, продукт стал непригодным для изучения, непригодным для применения с точки зрения производительности и понимания, имел низкую надежность и постоянно уничтожал данные. Забитый доверху «уникальными» возможностями, продукт не обладал не необходимыми базовыми функциями, существовавшими в основе всех конкурирующих продуктов.

Как можно было предположить, к концу февраля совет директоров принял меры, и президент с вице-президентом по разработке были вынуждены оставить свои посты.

Конечно, это всего лишь один эпизод. И он мог бы показаться частным случаем, если бы не повторялся многократно в компаниях, где мне приходилось работать за последние двадцать с лишним лет.

По моему наблюдению, от продукта можно добиться только тех свойств, которые были учтены заранее. Все, что мы имели до января, – это только сроки и обещанные функции. Не было никаких требований к качеству (среднему времени между сбоями, повреждениями данных и т. д.), поэтому качество было принесено в жертву. Не было метрик оценки производительности (сколько секунд должно пройти между нажатием на клавишу и появлением результата), поэтому паузы в реакциях системы получили произвольную длину. Не было никаких представлений о том, сколько времени должно занять изучение функции или насколько часто пользователь будет работать без ошибок, поэтому в жертву были принесены простота изучения и эргономика. Но цели, подвергшиеся оценке (сроки сдачи и перечень возможностей), были достигнуты, а поскольку полного описания функций также не было, многие из них были достигнуты лишь номинально.

Скотт подчеркивает фундаментальную истину: «Что посеешь, то и пожнешь». Если все время смотреть на календарь, то проект будет сдан вовремя, а если вам все равно, будет ли пользователь удовлетворен продуктом, то о пользователя просто вытрут ноги. Скотт продолжает рассказ:

Инвесторы не устают повторять: «У нас не так много денег, чтобы тратить их на продукт, который мы не сможем продать, поэтому мы должны изучить покупателей, прежде чем начнем проект». И при этом руководители разработки, похоже, неизменно верят в то, что у нас не так много времени и денег, чтобы тратить их на проектирование взаимодействия. Мы можем проектировать до бесконечности, и деньги закончатся прежде, чем мы успеем сделать продукт». Поэтому в конечном итоге они создают новые и новые продукты, вместо того чтобы совершенствовать уже имеющиеся – пока не закончатся деньги…

Если посмотреть со стороны, последние несколько месяцев были похожи на старую эксцентричную комедию или мыльную оперу, только без всякой романтики. Тоже по-своему ценный опыт. Конечно, смотря на дело только так, история не стоила бы столь подробного рассказа. Но во мне говорит сильное чувство. Думаю, долг чести требует прекратить впустую тратить жизнь людей на столь бесполезные предприятия.

В книге «About Face» ты писал о том, как важно перестать тратить время пользователя. От всего сердца соглашаюсь. Но это лишь вершина айсберга. Время пользователя можно потратить только тогда, когда продукт, достигший рынка, начинают покупать. Многие проекты закрываются прежде, чем разработчики успеют создать продукт, а результатом многих становятся продукты, не находящие покупателей. Инженерам из числа знакомых мне далеко не безразлична судьба их продуктов. Однако когда проект закрывают или продукт получается неудачным из-за отсутствия проектирования, то это означает пустую трату их энергии. А в мире не так уж много квалифицированных кадров, чтобы впустую тратить их время. Долг чести, о котором я говорю, призывает не просто «перестать впустую, тратить время пользователей, но перестать впустую тратить время и жизни всех людей, включая программистов.

Было очень больно оказаться Кассандрой в роли наблюдателя – предсказывать мрачную судьбу и, находясь в роли наблюдателя, смотреть, как мимо проплывают возможности ей воспрепятствовать. Я пришел к заключению, что обучение методом проб и ошибок оказывает воздействие настолько сильное, что прошедший такое обучение человек становится глухим для доводов, основанных на фактах и цифрах.

Хочу подчеркнуть, что опыт Скотта вполне типичен. Вот история другого коллеги из нашей области, Джона Ривлина (John Rivlin). Джон управляет небольшой, но очень успешной компанией в Пало-Альто, специализирующейся на проектировании и разработке программного обеспечения. Он прислал мне свою историю:

> Мы всегда тщательно проектировали продукты, прежде чем начинать разработку, и данный конкретный случай – не исключение. Мы начали проект с создания пятнадцатистраничной спецификации, описывающей взаимодействие пользователя с программой, которую мы предлагали написать. Спецификация включала и общие проектные положения, что дало нам возможность выйти за пределы начального описания «в одно предложение». Это важно, поскольку мы работаем исходя из фиксированной стоимости разработки и идем на определенный риск.
>
> Руководитель разработки нашего клиента, управляющий проектом, поддержал идею написания спецификации, и мы согласовали фиксированную цену. После этого готовую спецификацию передали боссу руководителя разработки, техническому директору. Вот какой ответ мы получили от него: «Зачем вы потратили так много времени на спецификацию? Вы потратили серьезную часть проектного бюджета. Мы не занимаемся спецификациями. Мы просто берем и делаем работу». При дальнейшем разбирательстве выяснилось, что представления технического директора о функциональности существенно отличались от представлений руководителя разработки. Несоответствие выявилось лишь благодаря «бесполезной спецификации», однако и этот факт не убедил его в пользе проектирования программного обеспечения. И это технический директор технологической компании, акции которой находятся в свободном обращении, а ежегодные прибыли превышают сто миллионов долларов. Вот уж, воистину, психбольницей управляют пациенты.

Страх перед проектированием, живущий во многих руководителях разработки, иррационален, но часто базируется на вполне реальном личном опыте. В прошлом, в поисках более совершенных продуктов, руководители просили программистов проектировать взаимодействие, и результаты оказывались плачевными. Человек, не владеющий методами проектирования взаимодействия, стремится к созданию продукта, пользователем которого является сам, и программисты, разумеется, тоже попадают в эту ловушку. Любая группа людей, соотносящая будущий продукт с собой, будет бесконечно долго тянуть резину, пытаясь придти к общему мнению по вопросам проектирования, потому что ее участники не имеют единого понятия о пользователях продукта.

Компьютеры против людей

Программное обеспечение больше похоже на мост, чем на здание. Программы работают на высокотехнологичных микропроцессорах, а управляют ими и используют их простые смертные. Осваивая новые технологии и восхищаясь ими, мы упускаем из виду огромную разницу между компьютерами и людьми, эти компьютеры использующими.

К примеру, мы считаем, что раз у компьютеров есть память, она должна быть похожа на человеческую. Это совершенно не так. Память компьютера работает иначе. Моя память позволяет мне легко распознавать лица друзей, а мой собственный компьютер никогда не узнает даже меня. Мой компьютер хранит миллионы телефонных номеров с идеальной точностью, а я не всегда сразу вспоминаю даже собственный номер.

Чтобы программное обеспечение было мощным и надежным, оно должно создаваться в идеальной гармонии с потребностями кремниевых микросхем. Чтобы программисты работали профессионально, они также должны участвовать в этой гармонии.

Чтобы пользователи были довольны и могли эффективно применять программы,

программы следует создавать в гармонии с потребностями человеческой природы. Проблема, понятно, в том, что человеческие потребности радикальным образом отличаются от потребностей кремниевых микросхем.

Очевидно, что одна часть программы, а именно внутренняя, должна создаваться с применением специальных технических знаний и учетом потребностей компьютеров. И точно так же очевидно, что вторая часть (внешняя) должна создаваться с применением специальных социальных знаний и учетом потребностей людей. Я убежден, что программисты способны справиться с первой задачей, но вторая задача требует участия проектировщиков взаимодействий.

Идеолог компьютерной отрасли Джерри Вайнберг говорит:

> «Найдя решение главной проблемы, вы делаете главной проблемой следующую по списку»[17].

Многие десятилетия главной проблемой компьютерной отрасли оставалась эффективность. Компьютеры были, в основном, маленькими, дорогими, медленными и непроизводительными. Мы обожествляли хакеров, умевших создавать максимально эффективные программы и выжимать всю возможную производительность из дорогих ЭВМ. По существу, было гораздо дешевле обучать людей общению с непонятными, но производительными компьютерными программами, чем покупать дополнительные компьютеры. Неуклонное и неизбежное падение цен на компьютеры навсегда избавило нас от этой проблемы. Сейчас, оказывается, гораздо дороже обходится адаптация людей к «эффективным» программам, чем создание программ, соответствующих ожиданиям людей.

Решение очевидно: поставить программы на службу пользователям. Однако на пути такого решения встает культура, которую мы столь заботливо создавали последние сорок лет, культура, обожествляющая хакеров, стоящих у руля. Сообщество разработчиков программного обеспечения в целом готово включить проектирование взаимодействия в процесс разработки. «Конечно, – говорят они, – проектируйте сколько угодно, только дайте сначала продукт закончить». К сожалению, проектирование взаимодействия должно предшествовать строительству, поэтому открытость программиста проектированию совершенно бесполезна. Точно так же оператор бетономешалки может сказать плотникам, что они смогут начать создавать каркас, как только он закончит заливать бетон.

Учим собак быть кошками

В качестве отступного разработчики программного обеспечения всегда выражают готовность изучать проектирование. Меня постоянно просят «научить проектировать». Я приветствую такую открытость, но не верю в эффективность такого подхода. Любой разработчик программного обеспечения, достаточно квалифицированный, чтобы называться профессионалом, слишком погружен в буквальную и детерминированную сущность

[17] Gerald Weinberg, «The Secrets of Consulting: A Guide to Giving & Getting Advices Successfully» (Секреты консультирования: Руководство по успешному применению советов), Dorset House, 1985.

кремниевой логики. Слишком сильно погружен, чтобы достичь параллельно схожей эффективности в иррациональном, непредсказуемом, переполненном эмоциями мире людей. Я не хочу сказать, что программист неспособен стать проектировщиком, я лишь пытаюсь сказать, что практически невозможно делать то и другое хорошо одновременно.

Каждый разработчик программного обеспечения считает себя не таким, как все, считает, что способен делать и то и другое. Как ясно показала судьба General Magic, это попросту не так. Разработку в General Magic возглавляли Билл Аткинсон (Bill Atkinson) и Энди Герцфельд (Andy Herzfeld), в прошлом ведущие разработчики программного обеспечения для Apple Macintosh и, вероятно, два наиболее изобретательных и одаренных творца из числа программистов. Их совместное программирование проектирование для Macintosh превратилось в успех в 1984 году (хотя в проектирование пользовательского интерфейса существенный вклад внес Джеф Раскин (Jef Raskin), в программировании участия не принимавший). Однако за последующие четырнадцать лет вещи достаточно сильно изменились, и старые методы перестали быть применимыми. В начале 1993 года я брал интервью у Энди Герцфельда в штаб-квартире разработки General Magic, которая являлась одновременно и его жильем в Пало-Альто. Там он изложил мне свою философию проектирования программных продуктов. Я изумленно слушал его, понимая, насколько малы его шансы на успех. История признала выдающийся талант Энди.

Несомненно, что продукт, задуманный General Magic, был востребован, и таковым остается поныне. Несомненно, что технология в продукте применялась великолепная. Несомненно, что способность Марка Пората находить стратегических партнеров и заключать сделки мало кому удастся превзойти. Несомненно, что компания имела хорошую родословную и хорошее финансирование. Что же тогда уничтожило компанию? Я считаю главной причиной проектирование взаимодействия, а точнее – его отсутствие. Несмотря на звездную генеалогию и вселяющие трепет таланты, продукт General Magic был *сконструирован,* а не *спроектирован*.

Принятый в отрасли образ мышления не дает места столь очевидным выводам, как видно из статьи о General Magic. Чаша ответственности за провал продукта в статье склоняется, похоже, к высокомерию и честолюбию Пората, однако в Кремниевой Долине нет ни одного президента, у которого имеется недостаток таких проявлений собственного эго. Эти качества навряд ли могут быть причиной провала компании.

Наша высокотехнологическая культура настолько замкнута в себе, что мы слабо осознаем собственные провалы и слабые места. Невозможно стать успешным журналистом в этой области, не будучи компьютерным фанатиком – апологетом, поэтому журналисты сваливают вину за провалы на наши личные качества, недоброжелательность фортуны и форс-мажорные обстоятельства.

* * *

Разработка программного обеспечения не является самостоятельной профессией, такой как юриспруденция, архитектура или медицина, поэтому названия должностей в нашей отрасли весьма ненадежны. Некоторые мои друзья, высокопрофессиональные программисты, называют себя «проектировщиками программного обеспечения». Самоназвание, вне всякого сомнения, заслуженное, но не совсем верное. Скажем, Энди Герцфельд с готовностью отзывается на прозвище «проектировщик».

Многие программисты считают себя талантливыми проектировщиками. Вообще говоря, это действительно так во многих случаях, но существует огромная разница между проектированием *функциональности* и проектированием для *людей.*

Даже если программистов сложно оправдать в плане проектирования, они, по крайней мере, стараются удерживать проекты от окончательного расползания. Завидев узурпатора, они стараются не допускать к рулю безответственных людей. Большинство программистов очень ответственны, они часто считают сторонних консультантов, маркетологов и

руководителей вздорными и некомпетентными личностями.

Программисты чувствуют чепуху за версту, и всего после пары случаев, когда маркетологи или руководители требуют «изменений, улучшающих интерфейс» и оказывающихся в итоге неэффективными, у программистов возникают защитные барьеры против такого вмешательства. Изменения могут быть к лучшему или к худшему, но в любом случае программисты вынуждены делать дополнительную работу. Каждое изменение снижает качество кода, поскольку неизбежно оставляет швы и рубцы. Если кто-то заявляет, что программой станет легче пользоваться, и достаточно лишь поместить каждую кнопку «ОК» в правый верхний угол диалогового окна, опыт и мудрость программиста подсказывают ему, что это пустая трата времени – *его времени.* И он прав в своей бдительности.

После нескольких напрасных погонь за недостижимыми целями они начинают относиться к поступающим извне рекомендациям по проектированию, как к обычным советам. Они словно строители, которым пришлось разрушить слишком много непродуманных стен и которые теперь смотрят на чертежи с предубеждением и зарекаются воспринимать их всерьез.

* * *

Разработчики программного обеспечения рисуют на досках диаграммы, изображая пользовательский интерфейс и код для работы с данными в виде отдельных прямоугольников. Однако реальной разницы в кодировании того и другого нет. Это не две стены, одна из которых сложена из гранитных блоков квалифицированным каменщиком, а соседняя – из деревянных досок, сколоченных плотниками и обшитых гипсовыми изоляционными панелями. Совсем нет. В коде, реагирующем на движения мыши, и коде, реорганизующем базу данных глубоко в недрах программы, используются примерно те же операторы, указатели, вызовы методов. Часто внутренний код системы и код для взаимодействия с пользователем пишет один и тот же человек. Он пользуется одним языком, теми же библиотеками, инструментами и методами для решения этих задач. Кто может сказать, где проходит граница между программой для компьютеров и программой для пользователей?

Программисты привыкли рассматривать задачи в рамках функций, так что им совершенно неясно, чем хороша идея взять фрагмент программы, нарушить множество границ функциональности и перенести его на сторону пользователя. Инженерам трудно осознать, чем код на языке C, реализующий взаимодействие с базой данных, разительно отличается от кода на языке C, реализующего взаимодействие с человеком.

* * *

Следующую историю рассказал мой коллега, Джим Гей (Jim Gay). Он показывает, как легко умные инженеры загораются увлекательными и интересными проблемами, вместо того чтобы заниматься решением проблем, действительно того требующих.

> Начинающая компания TransPhone решила выйти на рынок электронной коммерции. Основной нашей идеей стало создание простого в применении смартфона, как основы для интернет-коммерции. Решающим фактором успеха нашей модели был простой интерфейс, с которым было бы удобно работать людям, не имеющим отношения к компьютерам. TransPhone обратилась за помощью в компанию, специализирующуюся на проектировании взаимодействия.
>
> Мы считали, что практически уже создали пользовательский интерфейс, однако ему не помешала бы некоторая доводка. На первом же собрании проектировщики взаимодействия много раз повторили, что понятия не имеют, что мы в действительности пытаемся создать или для кого мы пытаемся это создать.

Мы посчитали, что они поверхностно смотрят на проблему, которая на самом деле была довольно серьезной. Собрание закончилось тем, что проектировщики попросили нас более точно определить цели и задачи. У нас появилось ощущение, что эти проектировщики ни малейшего представления не имеют о том, чего мы пытаемся достичь.

После этого мы создали улучшенный прототип для демонстрации потенциальным партнерам, однако устройство TransPhone попросту не вызвало у них восторга. Мы продолжали считать, что демонстрация просто недостаточно убедительна. TransPhone прекратила свое существование через несколько недель после создания второго прототипа.

Вспоминая то, первое собрание с участием проектировщиков взаимодействий, я отчетливо понимаю, что главную нашу проблему они обнаружили в первые же несколько минут: какова была наша цель, для кого мы все это делали? Никто и никогда не дал адекватного ответа на этот вопрос. Задайся мы сразу этим вопросом, возможно, случилось бы одно из двух: найдя ответ, мы получили бы шансы на успех либо, не найдя ответ, минимизировали потери инвестора.

Каков урок этой истории? Проектирование продукта является важнейшей составляющей жизненного цикла предприятия. Наша неспособность разрешить фундаментальные вопросы проектирования и желание вместо этого устремиться вперед, к разработке и продажам, в конечном итоге оказались для компании роковыми. Теперь-то понятно, что, не сумев создать представления того, что мы пытались делать, мы должны были вернуться к исходным целям своего предприятия. Думаю, это, скорее всего, привело бы к созданию иного, более простого продукта. Вместо этого мы продолжали добавлять прибамбасы, вероятно, еще сильнее маскируя возможную полезность продукта.

Подобно ребятам из General Magic, Джим на горьком опыте убедился, что, классная технология и раскаленный докрасна рынок не способны поднять тяжелый груз плохо продуманного кода. Недостаточно перебросить мост между технологией и потребностью. Кто-то еще должен сделать так, чтобы люди захотели ходить по этому мосту.

История наших технологий относится преимущественно к индустриальному веку, поэтому проблемы и решения, присущие ей, близки современному человеку. Сила сопротивления между людьми и механическими устройствами существует, но в определенных рамках. В информационную эру нашу жизнь заполонили компьютеры, все больше продуктов, содержащих микросхемы, и мы обнаруживаем, что между людьми и устройствами возникает когнитивное сопротивление, с которым мы не готовы бороться. Наши инженерные таланты высокосовершенны, но подводят нас в решении проблемы когнитивного сопротивления. За многие годы разработчики программного обеспечения определенно выросли как профессиональны, однако их способность создавать мощные и приятные программы остается такой же слабой, как и прежде.

Я считаю, что наша неспособность решить проблему инженерными методами доказывает невозможность ее решения таким способом. Более того, осмелюсь утверждать, что инженерные методы как раз и являются одной из причин возникновения, этой проблемы. Просить инженеров исправить ситуацию – все равно, что просить лису защитить курятник.

Глава 7
Homo Logicus

С большой долей иронии я называю программистов *хомо логикус.* Вид хомо логикус слегка – но достаточно ощутимо – отличается от вида *хомо сапиенс,* человека разумного. Из собственных наблюдений я почерпнул четыре фундаментальных отличия образа мысли и действия разработчиков программ от обычных людей. Об этих отличиях и пойдет речь в

данной главе. Программисты пожертвуют простотой ради контроля. Обменяют успех на понимание. Они сосредотачиваются на исключительных ситуациях вместо того, чтобы сосредоточиться на типичных. И, наконец, ведут себя грубо и прямолинейно, как быки.

Авиационный тест

Чтобы подчеркнуть различие между видами, я применяю забавную лакмусовую бумажку – «Авиационный тест». Чтобы пройти тест, достаточно представить, что вы идете по посадочному коридору авиалайнера. Вступив на борт, вы должны выбрать – пойти налево в кабину или же направо в салон.

В кабине пилота целый калейдоскоп сложных органов управления и счетчиков, на всех поверхностях располагаются датчики, ручки и тумблеры. Справа же, в салоне, полная противоположность – мягкие округлые формы, успокаивающие бежевые оттенки.

Если повернуть налево, в кабину, вам придется в совершенстве овладеть сложными техническими материями. Придется узнать, зачем нужен каждый из приборов. За понимание всего этого нагромождения вы получаете ощущение определенного *контроля над ситуацией* и ответственность за посадку в нужном месте.

Свернув направо, в салон, вы отказываетесь от какого-либо влияния на полет. За отказ от контроля вы получаете возможность расслабиться, зная, что попадете в нужное место, и при этом самой сложной операцией будет включение и выключение лампы для чтения.

Авиационный тест четко делит человеческую расу на две категории: свернувшие налево стремятся *контролировать ситуацию* и *понимать,* как работает технология, а свернувшие направо стремятся *упростить* свои размышления и испытывать уверенность в *успешности* полета. Программисты – *хомо логикус* – всегда выбирают поворот налево. Пользователи *хомо сапиенс* – всегда выбирают поворот направо.

Психология программистов

Поскольку наша цель есть создание продуктов, основанных на программном обеспечении, мощных и приятных для пользователей, понимание психологии пользователя может показаться естественным условием. Это, разумеется, верно, но мешает уловить более важное, но далеко не столь очевидное обстоятельство. Найти решение и реализовать решение – два совершенно разных действия. Я бы предпочел иметь в руках недостаточно хорошо спроектированный продукт, чем спецификацию великолепного проекта, пылящуюся на полке. Чтобы наши толком спроектированные продукты попали в руки пользователей, необходимо выполнить еще более важное требование: понять психологию создателей продукта, программистов.

Ничего не изменится, пока мы не начнем влиять на разработчиков программного обеспечения. Даже если программисты соглашаются с тем, что к пользователям следует относиться лучше (а обычно они соглашаются), это не означает, что они сделают все необходимое, чтобы достичь озвученной цели. Вы не заставите их изменить подход простыми просьбами. Чтобы получить работающее решение, мы должны проникнуть в способ их мышления, понять, как можно *мотивировать* этих людей на создание взаимодействия, удобного для пользователя. Разумеется, проектировщик взаимодействия должен разбираться в психологии, причем не только в психологии пользователей, но и в психологии разработчиков программ.

Смысл изложенного прост: программисты отличаются от обычных людей. Стереотипы их поведения вот уже много лет являются поводом для шуток: неловкость в общении, карманные предохранители[18], педантичность. Это лишь поверхностные признаки, их легко

[18] Речь идет о пакетиках, носимых в нагрудных карманах и предохраняющих рубашку от порчи в случае, если потечет ручка. Такие предохранители стали одним из символов культуры «ботаников» в начале второй половины XX века. – *Прим. перев.*

заметить и высмеять. По-настоящему существенные отличия не только гораздо тоньше, они способствуют более заметному увеличению когнитивного сопротивления в создаваемых программистами интерактивных продуктах.

Многие обозреватели компьютерной индустрии приложили усилия, чтобы определить эти отличия. Роберт Кринджели (Robert Cringely) называет программистов «смердящими богами», подразумевая одновременное высокомерное отношение к окружающим и личное отношение к гигиене.

Другой проницательный наблюдатель и талантливый автор – По Бронсон (Po Bronson). Он обращал свое зоркое око и острый ум к миру высоких технологий. Пародируя Стивена Кови (Steven Covey), он создал список «Семь привычек крутых инженеров». Эти определения невероятно точны, хотя и гиперболичны.

1. Они щедры в своем эгоизме.
2. Слепота улучшает их зрение.
3. Они кусают не только руку кормящего, но еще и собственные руки.
4. Они готовы приложить любые усилия, чтобы сохранить впечатление, будто их не заботит собственный имидж.
5. Они чинят то, что не сломано, до тех пор, пока это не сломается.
6. «Не я дал неверный ответ, а вы задали не тот вопрос».
7. Считают отсутствие критики комплиментом.

Программисты пожертвуют простотой ради контроля

Хомо логикус желает контролировать то, что его интересует, а интересуют его сложные, детерминированные системы. Люди тоже сложны, но их поведение нелогично и труднопредсказуемо, они не ведут себя, как механизмы. Лучшие механизмы – это цифровые механизмы, поскольку они самые сложные, самые изощренные, и программист без труда может их модифицировать.

С точки зрения программиста контроль над людьми менее привлекателен. В своем романе «The First $20 Million Is Always the Hardest»[19] (Тяжелее всего даются первые двадцать миллионов долларов) По Бронсон (Po Bronson) заставил программистов устраивать розыгрыши над людьми, чтобы показать свою власть, однако программистам намного больше нравится повелевать компьютерами.

За контроль всегда приходится платить – дополнительными усилиями и увеличением сложности. Большинству людей по плечу разумные усилия, однако, программистов от большинства обычных людей отличает готовность и способность овладевать крайне сложными вещами. Понимать сложные системы, составленные из многочисленных взаимодействующих факторов, управлять такими системами – вот часть работы программиста, приносящая ему удовлетворение. Пилотирование самолета – увлечение для программистов[20]. Панель управления в кабине самолета просто забита индикаторами, ручками и рычагами, однако программисты преуспевают в общении со столь устрашающе сложными объектами. Для *хомо логикус* это веселое и увлекательное занятие, несмотря на необходимость (благодаря ей!) многие месяцы кропотливо учиться. *Хомо сапиенс* предпочитает лететь в пассажирском салоне.

Для *хомо логикус* контроль – цель, тогда как сложность – просто цена, которую они

[19] Po Bronson «The First $20 Million Is Always The Hardest», Avon Books, New York, 1997.

[20] Ладно, сознаюсь: я пилот. В 1979 году типичный программист-фанатик Гари Килдалл взял меня на борт своего Piper Archer. Этот короткий полет посадил меня на иглу авиаполетов. Компьютерный программист во мне любит всю эту бессмысленную сложность.

готовы платить за достижение цели. Для нормальных людей цель – это простота, и отказ от контроля – цена, которую они готовы платить. В продуктах, основанных на программном обеспечении, контролем являются функции. К примеру, в Windows 95 функция «Поиск файла» дает мне серьезный контроль над ходом работы. Я могу указать, в каких областях диска следует выполнять поиск, тип искомого файла, искать ли файл по имени или по содержимому и еще целый ряд параметров. С точки зрения программиста все это просто замечательно. Затратив дополнительные усилия и приобретя определенное понимание предмета, он получает возможность искать быстрее и эффективнее. Напротив, с точки зрения пользователя все не так радужно, поскольку ему *приходится* указывать область поиска, указывать тип искомого файла и еще метод поиска.

Хомо сапиенс с радостью принесли бы в жертву лишнюю минуту компьютерного времени, если бы знали, что не придется изучать, как работает функция поиска. Для них каждый параметр поиска – дополнительная возможность сделать что-нибудь не так. Вероятность ошибки и отрицательных результатов поиска возрастает по мере увеличения гибкости. Они бы с радостью принесли в жертву всю эту ненужную сложность, контроль и понимание процесса, чтобы упростить себе работу.

Программисты обменяют успех на понимание

Хомо логикус не может устоять перед тягой познания – он просто обязав узнать, как работают вещи. *Хомо сапиенс* же, напротив, стремится к достижению успеха. Программистам знакомо желание добиться успеха, но они часто принимают провал как цену, уплаченную за понимание.

Есть старый анекдот об инженерах, дающий некоторое представление о потребности понимать.

Три человека приговорены к казни: священник, адвокат, инженер. Первым на эшафот вступает священник. Палач дергает за рычаг, открывающий люк, но ничего не происходит. Священник заявляет, что это божественное вмешательство, и требует, чтобы его освободили, что и происходит. Выводят адвоката. Палач дергает за рычаг, и снова осечка. Адвокат заявляет, что еще одна попытка будет расценена как повторное привлечение к ответственности за то же преступление, и требует, чтобы его отпустили, – что и происходит. Наступает очередь инженера, который приступает к внимательному изучению механизма виселицы. Прежде чем палач успевает дернуть за рычаг, инженер поднимает взгляд и говорит: «А я понял, почему она не работает».

Понимание механизма работы виселицы оказалось интереснее собственной жизни.

Читая лекции компьютерным программистам, я прошу поднять руки тех, кто в детстве разбирал часы, чтобы посмотреть, как они работают. Как правило, две трети присутствующих поднимают руки. Затем я спрашиваю, скольким из них удалось в конечном итоге собрать часы, и большая часть рук опускается. Мой следующий вопрос таков: кто из вас считает этот эксперимент неудавшимся? Большая часть присутствующих смеется, осознав, что получили удовольствие от разрушения часового механизма. *Хомо логикус* желает понять, как работают часы, – такова его цель, и он вполне готов принести в жертву работающие часы, чтобы этой цели достигнуть. С другой стороны, *хомо сапиенсу* нравится, когда часы работают. Его цель состоит в том, чтобы узнать, который час, и взамен он отказывается от знания о том, что заставляет часы тикать.

Проектировщик Джонатан Корман отмечает:

Большинство людей не понимают, до какой степени компьютеры захватывают программистов. Сложности изучения компьютеров лишь усиливают в программистах чувство удовлетворения. Их интерес настолько искренний и глубокий, что им никогда и в голову не приходит, что другие могут чувствовать что-то иное, а потому причиной раздражения других людей они считают неспособность к обучению, но никак не отсутствие интереса.

Тяга программистов к пониманию заставляет их инстинктивно создавать взаимодействие, приближенное к внутренним механизмам продукта. Вместо того, чтобы делать программы, отражающие конечные цели пользователей, они отражают работу внутреннего механизма программы. Естественно, что программисты не испытывают неудобств, пользуясь такими программами, поскольку, понимая принцип работы программы, они способны понять и способы ее применения. Мы называем этот распространенный стиль взаимодействий «моделью реализации». К примеру, компьютерные документы постоянно хранятся на дисках, однако программы способны модифицировать только документы, временно загруженные в оперативную память. Программистам весьма комфортно с такими техническими нюансами, поэтому интерфейсы их программ отражают оба типа присутствующей в компьютере памяти. Для пользователя же подобные вещи аналогичны абсолютно неуместному на приборной доске автомобиля переключателю, заставляющему выбирать между шинами с радиальным и диагональным кордом.

Нормальные люди вполне согласны не иметь представления о работе предмета, даже если применяют предмет постоянно и никак иначе прожить не могут. Они считают, что интерфейсы, созданные по модели реализации, возлагают на них ненужное бремя понимания. Программистам подобное отношение кажется непостижимым.

Программисты сосредотачиваются на исключительных ситуациях

Программисты разделяют присущий математикам взгляд на сложные системы, а потому неудивительно, что они смотрят на вещи не так, как большинство людей. Что я имею в виду? Представьте, что вы подбросили монету 1000000 раз; из них 999999 раз монета упала орлом вверх, и только один раз вверх решкой. Для математика утверждение «монета всегда падает орлом вверх» является ложным. Единственный раз, когда монета упала вверх решкой, опровергает утверждение. Говоря математическим языком, утверждение верно, если оно верно *всегда.* Этот образ мысли привычен и кажется разумным *хомо логикус,* поскольку именно так ведут себя компьютеры.

С другой стороны, нормальные люди в большинстве своем, столкнувшись с приведенным утверждением, заявят, что оно истинно, исходя из преобладания событий, подтверждающих это предположение. Кроме того, они заявят, что утверждение не просто верно, но верно всеподавляюще, убедительно, неоспоримо. Вероятность того, что оно верно – миллион к одному! В контексте человеческого поведения ставки миллион к одному трактуются однозначно. Это вероятность, которую нет смысла оспаривать и обдумывать. Шансы, что меня ударит молния, что я случайно упаду с моста или выиграю в лотерею, больше, чем шансы, что монета упадет вверх решкой.

Вероятность правдивости утверждения о монете огромна, а *хомо сапиенс* живет в мире повторяющихся событий. Но всегда есть вероятность отрицательного результата, а программисты живут в мире возможностей. Если это может произойти, об этом *следует* задуматься. В мире программного обеспечения, где преобладают точно сформулированные утверждения, даже маловероятные события попросту нельзя игнорировать.

Программисты называют эти события с крайне низкой вероятностью «исключительными ситуациями»[21]. Наступление подобных событий маловероятно, но если их не предусмотреть, программа даст серьезный сбой, когда такое событие произойдет. Несмотря на низкую вероятность описываемых событий, за неподготовленность к оным приходится платить огромную цену. Таким образом, маловероятные события становятся для программистов вполне жизненными ситуациями. Тот факт, что граничные условия могут наступать лишь раз в 79 лет ежедневного применения программы, программиста совершенно не утешает. *Что если этот Раз наступит завтра?*

Есть основания полагать, что самым главным отличием профессионала от дилетанта в сфере программирования является одержимость эксперта подготовкой к исключительным ситуациям. Столь фанатичная подготовка к вероятному неизбежно заслоняет правдоподобное. Результатом становятся продукты, взаимодействие с которыми щедро сдобрено редко востребованными или совсем не востребованными органами управления, мешающими работать с нужными функциями. Самая распространенная жалоба пользователей: с программой тяжело работать, потому что в ее интерфейсе слишком много настроек, не отличающихся одна от другой.

Замечательный пример «щедрости в эгоизме», по Бронсону, – изобилие ненужных и нежелательных возможностей, появляющихся в результате возможностного мышления программистов. Они поставляют нам много такого, что нужно только *им самим*.

* * *

Программисты шутят, что существует всего три числа: нуль, единица, бесконечность. В мире компьютерных вычислений она обретает смысл. В двоичном внутреннем мире компьютера процесс либо происходит, либо нет – это нуль или единица. Если какое-либо событие может случиться единожды, это означает, что оно может повториться бесконечное количество раз.

Код установки программы и завершения работы системы пишется таким образом, что может выполняться лишь единожды. Если программа попытается выполнить его повторно, это может привести к сбою компьютера или, по меньшей мере, к серьезным ошибкам в работе. Другие фрагменты программы спроектированы таким образом, что могут выполняться повторно. Практически любая часть произвольной программы, *способная* отработать дважды без сбоев, может исполняться любое число раз. Для кода и для *хомо логикус,* программиста, нет особой разницы между двумя запусками и двумя миллионами или миллиардами запусков.

[21] Другие названия – «крайние случаи», «специальные случаи», «граничные условия».

А что же люди? Они понимают нули и единицы, но, кроме того, еще твердо понимают двойки, семерки и число тридцать один. Большинству людей сложнее представить миллион вещей, чем 300 вещей. Типичный человек выполняет действия в количествах, которые не исследуются программистами. Скажем, заядлые лыжники-любители могут потратить на лыжные походы десяток выходных за сезон. За сорок лет активного катания это составит менее 500 раз. Современные цифровые компьютеры способны обработать 500 объектов за долю секунды. Фанат любой программы запустит ее не более нескольких тысяч раз, а программисты продолжают думать в масштабах бесконечного числа событий.

Хорошие программисты преднамеренно игнорируют такие практичные числа, как 500, потому что это повышает готовность программ к возможному пятьсот первому разу. Именно это подразумевает По Бронсон, говоря, что «слепота улучшает их зрение».

Программисты ведут себя грубо и прямолинейно

Вероятно, самое удивительное в хороших программистах – это их подражание бугаям. Я вполне сознательно употребляю слово «бугаи», поскольку оно ассоциируется с незрелостью, эгоизмом, соперничеством, а также с физической силой и координацией.

Говоря «бугай», я вспоминаю уроки физкультуры в средней школе. Некоторым ребятам природа дала более развитую, сильную мускулатуру и хорошую координацию. Они добивались превосходных результатов в организованных занятиях спортом, однако, кроме того, обнаружили, что могут подчинять себе других, не таких сильных соучеников. Эти «спортсмены» господствовали не только на бейсбольных и футбольных полях, но еще и в раздевалках и на школьном дворе, в соревнованиях, не предусмотренных расписанием.

Семнадцатилетний парень ростом метр восемьдесят обладает силой мужчины, но не обладает его зрелостью. Этот мужчина-мальчик не проявляет сострадания к тем, кто слабее. Его донимают подростковые муки, и к нему еще не относятся как ко взрослому человеку. Его философия жестока и проста – держи удар или умри: «Не можешь сделать то же, что и я? Ты просто никчемный неудачник». Любой юноша на игровой площадке, не способный состязаться с ним в физическом плане, считается недостойным. Имея физическую возможность доминировать, он доминирует.

И вот с этим амбалом происходит интересная вещь. Окончив школу и попав в реальный мир, он обнаруживает, что способность физически доминировать над другим человеком быстро перестает быть его сильной стороной, становится бесполезной. В школе потенциальную угрозу со стороны круглолицего очкарика можно с легкостью устранить: пара хороших ударов и надменный гогот школьной спортивной команды быстро ставят парня на место. В деловом мире кулаки и нахальство использовать невозможно. В конференц-зале недопустимы и не действенны приемы с пинками и ударами мокрым полотенцем. И хотя бугай по-прежнему сохраняет физическую возможность для победы над другим, более слабым человеком, в случаях, когда этот человек является коллегой или руководителем, победа обязательно станет пирровой.

Бугаи, в прошлом незрелые школьники, обнаруживают, что приходится выучить весьма унизительный урок. Вырвавшись на просторы внешнего мира, они понимают, что общество обрезало им крылья, и учатся успешно сосуществовать с людьми, обладающими менее развитой мускулатурой. Бугаи широко представлены в бизнесе, и в конечном итоге они неплохо справляются. Они успешно совершают этот переход, пусть без готовности и желания. Сохраняя врожденный дух соревнования, они достигают также определенного уровня зрелости и самоотверженности, становясь достойными членами общества.

Программисты – совсем как эти бугаи. Учась в школе, многие программисты не имели физического уровня бугаев, но обладали более острым умом и лучшей координацией интеллектуальных функций. Они превосходно проявляли себя в организованной деятельности: в дискуссиях, в литературных клубах, в шахматной команде.

Что касается терзаний переходного возраста, здесь их способности стоили не так много, как мускулатура. На школьной спортплощадке более сильные юноши легко доминировали над ними. Тощий семнадцатилетний, обладающий познаниями взрослого в математике, физике и информатике, по-прежнему остается физически слабым мальчиком, которого игнорируют на футбольном поле и считают неудачником на любовном фронте. Этого ребенка мы называем «ботаником».

Он не проявляет сочувствия к тем, кто интеллектуально слабее, чем он. Про себя, не имея физической силы, чтобы делать это публично, он смеется над более крупными ребятами, не обладающими его смекалкой и интеллектом. Его философия проста и жестока: держи удар или умри. Любой другой присутствующий на «спортплощадке», не способный с ним состязаться, считается недостойным. Он не задумывается о чувствах или талантах более слабых людей. Система его ценностей выражена в неофициальной иерархии, основанной на внутреннем развитии его собственного острого интеллекта. В среде равных себе, не бугаев, его отношение таково: если я могу одолеть тебя – в интеллектуальном состязании – я твой повелитель и я лучше тебя.

Подобно бугаям, одаренным физической силой, хорошие программисты также от природы талантливы, и дух состязания в них не менее силен, чем в любом молодом спортсмене. Обнаружить этот дух может быть сложнее, поскольку программирование – «спорт» одиночный и в основном невидимый. Но пусть вас не обманут их тихие манеры, программисты серьезные конкуренты, а действительно хорошие программисты – головорезы не хуже будущих олимпийских чемпионов.

И вот с таким «ботаником» происходит интересная вещь. Перейдя из школы в реальный мир, он обнаруживает, что способность интеллектуально доминировать над другим человеком переходит нетронутой в зрелое, цивилизованное общество взрослых. Его здесь защищают социальные ограничения, и его уже не побьют на игровом поле. При переходе из подросткового мира в мир взрослых физическая сила перестает быть аргументом, но интеллектуальные разборки становятся все более сильным оружием.

Характер интеллектуального бугая – способность доминировать над другими при помощи умственных способностей – в мире взрослых информационной эры приобретает невиданную силу. В гражданском обществе стало допустимым применение интеллектуальных пинков в виде непостижимого программного обеспечения, стали допустимыми щелчки эмоциональными полотенцами по исстрадавшимся людям, которые всего лишь пытаются извлечь наличные из банкомата.

Бугаи, обладавшие огромной властью в школе, обнаруживают, что находятся теперь во власти своих бывших жертв. Процесс превращения во взрослых делает многих бугаев достойными людьми, и многие из них говорили мне с сожалением о своем подростковом поведении.

Бывший лучший центровой команды по баскетболу, ростом метр девяносто, обнаруживает, что его физическая доблесть бесполезна в зале для совещаний, тогда как бывший председатель школьного клуба астрономии ростом метр семьдесят пять обнаруживает, что его умственные способности позволяют маневрировать и наносить удары с непревзойденным проворством. Инфантильный «ботаник»-адвокат способен одержать победу в суде при помощи острого языка и изощренного ума. «Ботаник»-доктор получает власть над жизнью своих пациентов, в прошлом бугаев. И – вот это сюрприз – чахлый «ботаник»-программист получает доступ к огромной власти, поскольку теперь он контролирует доступ всех остальных людей к информации.

Применение этой силы не ограничивается никакими процессами взросления. Они доминируют над другими при помощи своих интеллектуальных способностей лишь потому, что могут это делать, и не видят ничего дурного в издевательствах над пользователями, каковыми являются устрашающе сложные продукты. Они глумятся, подшучивают, смеются над «чайниками», которые недостаточно умны, чтобы пользоваться компьютерами. Да и рабочие привычки этих людей – изоляция, многочисленные сверхурочные и внеурочные – не оказывают особого цивилизующего воздействия на их поведение.

Лишь приближаясь к тридцати годам, я осознал, каким был грубияном. Обычным грубияном, кулаками которого были способности к программированию, а ростом и длиной рук – владение сложными системами. И я издевался над теми, кому не по силам оказывались сложности использования компьютеров.

Глава 8
Отмирающая культура

Программирование – деятельность до некоторой степени «потусторонняя» и эмоционально весьма насыщенная. Именно такая насыщенность делает программирование более похожим на призвание, жаргон программистов более похожим на самостоятельный язык, и благодаря ей братство разработчиков программ создало свою культуру. В этой главе я покажу, каким образом культура программирования влияет на природу программных продуктов.

Культура программирования

В недавнем субботнем приложении к «Таймс» мне довелось прочесть занимательную историю американской пары, после выхода на пенсию уехавшей жить в Мексику. Они купили участок в предместье крупного города и наняли архитектора-американца для проектирования дома своей мечты. Затем они наняли мексиканского подрядчика и передали ему чертежи. В ходе строительства они с изумлением обнаружили, что получается совсем не тот дом, который описал архитектор.

Чертежи указывали, что на фасаде дома должно быть четыре окна, произведенные конкретным изготовителем, и приводили даже точный идентификационный номер товара. Владельцы дома обнаружили, что на фасаде три окна, совершенно иных по виду и размерам. На их расспросы мексиканский строитель пожал плечами и сказал: «Это ведь окна. В плане указано, что окна на этой стороне. В чем проблема?»

Владельцы и архитектор здания происходили из одной культуры, имели общие ценности, тогда как строитель происходил из другой культуры, а значит, иначе оценивал аспекты проблемы. Несомненно, ему удалось обеспечить заказчиков окнами за меньшие деньги и ценой меньших усилий, а это – в его мире – и было первоочередной задачей. Владельцы и архитектор, американцы, полагали, что наличие чертежей требует четкого следования этим чертежам. Строитель же, мексиканец, полагал, что чертежи – это лишь предложение, а не требование. Он полагал, что его собственные императивы бережливости и

простоты комплектования естественным образом преобладают над любыми спецификациями. Он искренне старался претворить в жизнь видение архитектора, но применял к проблеме собственные культурные фильтры – собственные ценности.

Повторное использование кода

Подобно мексиканскому строителю, который ставил стоимость выше соображений проектирования, предоставленные сами себе инженеры ценят эффективность программирования больше, чем это необходимо пользователю. Лучшим свидетельством в пользу этого тезиса является практика повторного использования кода, то есть кода, который был ранее создан для какого-то иного проекта или же мог быть приобретен за определенную сумму у сторонней фирмы. Написанный код не просто экономит время; очевидно, что и другие программисты могут его использовать, и к тому же код не содержит ошибок. Одно из уникальных свойств программ состоит в том, что любую процедуру можно выполнить всего одной командой, но при этом размер процедуры не ограничен. Иначе говоря, если процедура уже написана, ее можно задействовать одной командой. Следовательно, любой уже написанный модуль кода оказывается значительным подспорьем для программистов. Они могут включать его в свои программы в качестве черного ящика, внутреннее устройство которого никогда не требует их вмешательства. Программист таким способом экономит время не только непосредственно на этапе программирования, также и на размышлениях и тестировании. Для большинства программистов повторное использование кода становится более важным, чем практически любое другое соображение. Знаменитый идеолог UNIX Эрик Реймонд (Eric Raymond) говорит:

> «Хорошие программисты знают, что писать, великие – что надо использовать повторно».

Основной побочный эффект повторного использования кода заключается в том, что крупные блоки большинства программ существуют не потому что этого захотел некий проектировщик взаимодействия, но потому, что некий программист уже проделал необходимую работу в рамках другого бюджета. Многие программы из тех, что мы встречаем, существуют лишь потому, что уже существовали ранее.

К примеру, приложения для настольных компьютеров содержат так много меню и текстовых диалоговых окон потому, что все оконные системы Microsoft Windows, Mac OS, OS/2, Solaris – предоставляют готовые модули кода, обеспечивающие работу таких функций. И наоборот, ни одна из этих систем не содержит большого количества кода для механизмов drag-and-drop; вот почему в приложениях так мало непосредственного манипулирования (direct manipulation). Диалог можно создать при помощи шести-восьми строк простого декларативного кода. Идиома drag-and-drop требует примерно сотни строк весьма запутанного процедурного кода. Выбор программиста здесь очевиден. Выгоды конечного

пользователя в контексте такой экономии обычно не рассматриваются.

История с мексиканским строителем все время повторяется в разработке программного обеспечения, в основном благодаря стремлению программистов повторно использовать код. Эд Форман (Ed Forman), руководящий разработкой ПО в Elemental Software, прежде чем загрузить программистов работой, создает подробный и точный набросок внешнего вида экрана. Несмотря на это, говорит Эд, экран готовой программы являет собой лишь бледную тень этого эскиза.

Происходит это следующим образом. Набросок Эда содержит темно-серые кнопки на светло-сером фоне. Программист начинает создание интерфейса с копирования исходных текстов из другой, уже работающей части программы. Это хороший способ сэкономить время и силы в программировании, способ, очевидно, полезный для всех. Минус в том, что существующий код обрамляет кнопки дополнительной темно-серой линией. Кроме того, к темно-серой рамке крепится еще и текстовая подсказка. Вместо того, чтобы удалить текст и рамку в соответствии с наброском Эда, программист оставляет их, сохраняя множество строк кода. Код требует какой-то текстовой подсказки для каждой кнопки, поэтому программист вбивает некий подходящий, в его понимании, текст.

Увидев, наконец, программу с ненужными рамками и невнятными текстовыми подсказками, Эд изумленно качает головой. Когда он указывает на отличия программисту, тот просто не может понять, в чем проблема. Подобно мексиканскому строителю, программисты полагают, что их собственные императивы простоты создания и простоты комплектования готовым кодом имеют более высокий приоритет, чем чьи бы то ни было *предложения.*

Эда такое явление не только изумляет, но и раздражает, однако он не способен объяснить его. Его программисты, как один, сообразительны, способны, глубоко озабочены качеством продуктов и успехом компании, однако не могут устоять перед зовом сладкоголосых сирен. Разумеется, они постараются воплотить в жизнь задумки Эда, но не за счет собственных принципов реализации.

* * *

Привычка программистов повторно использовать код интересна их готовностью иметь дело с кодом сомнительного происхождения. Некоторые неопытные программисты начинают с черновых набросков первой попавшейся на глаза идеи, и этот фрагмент кода, будучи созданным, становится основой всех последующих усилий по разработке благодаря интенсивному повторному использованию.

Для примера: ядро операционной системы Windows создавали очень опытные программисты, а вот первые приложения, показывающие приемы взаимодействия программы с пользователем, были написаны практикантами и начинающими программистами той же Microsoft. Внутренний код Windows совершенствовался и переписывался, постепенно улучшаясь. При этом возмутительно большое количество популярных приложений до сих пор содержит длинные фрагменты кода, написанные двадцатилетними студентами, проводившими лето в Редмонде. То же верно и для Всемирной паутины. Экспериментаторы-дилетанты сварганили первые веб-сайты, а их последователи просто соорудили клоны этих первых сайтов, а их собственные сайты снова клонировали другие люди.

Как видите, существует явный конфликт интересов программистов и пользователей. Предвидя конфликты интересов в бесчисленных областях деятельности и профессиях, мы встраиваем защитные механизмы, ограничивающие влияние конфликта. Судьи и адвокаты имеют общие навыки, однако мы никогда не позволяем адвокатам выносить решения по делам. Мы никогда не позволяем баскетболистам судить собственные встречи. И вот налицо еще один конфликт интересов, однако, мы последовательно позволяем программистам принимать архитектурные решения, основанные на их личных предпочтениях.

В индустрии программного обеспечения и корпоративных компьютерных отделах широко распространено мнение, что именно программисты лучше всего подходят для проектирования программ, ведь они в этой области специалисты, обладающие наиболее глубокими познаниями в соответствующих вопросах. Кажется, вполне безобидным и естественным позволить программистам определять форму и поведение создаваемых ими программ, однако выбравшему этот путь не избежать ловушки конфликта интересов. Коварство этой ловушки не в различиях между программистом и пользователем, но в их сходстве. Пользователь желает достичь своих целей, а программист – своих. Проблема кроется в тонких различиях между этими целями.

* * *

Программисты настолько свыклись с повторным использованием кода, что часто копируют существующие методы, даже не копируя отдельные строки кода. Это происходит естественным образом, усугубляясь склонностью программистов к консерватизму. К примеру, большинство программ содержит многочисленные диалоги подтверждений, в основном ненужные. Многие из этих диалогов просто перекочевали из созданного ранее кода, но многие остались потому, что программисты привыкли включать их в код.

Недавно на конференции я встретил Джеффа Безоса (Jeff Bezos), основателя *Amazon.com,* и рассказал ему, как мне нравится интерфейс «в один щелчок» (1-Click) на его веб-сайте. Этот интерфейс позволяет посетителю заказать книгу – большой сюрприз – в один щелчок. Интерфейс действительно хорош, поскольку избавляет посетителя от докучающих деталей: можно просто нажать кнопку и не указывать повторно адрес и информацию по оплате.

Джеффри было приятно слышать, что мне понравился 1-Click, и он рассказал, что разработал идею со своими проектировщиками, после чего идею показали программистам. Те, разумеется, покивали и согласились, что задача решаема. Программисты удалились, и какое-то время писали код, а затем показали результаты Джеффу. Он нашел желаемую книгу и нажал кнопку 1-Click. И тут программа отобразила сообщение, требующее подтверждения! Программисты превратили интерфейс «в один щелчок» в интерфейс «в два щелчка». Для программистов это был лишь один дополнительный щелчок (делов-то?). Для Джеффа, как и для любого пользователя, это все равно что стопроцентный рост инфляции. Джеффу пришлось действовать кнутом и пряником, прежде чем программисты создали интерфейс 1-Click, позволяющий делать заказ в один щелчок. Джефф не скажет мне, насколько 1-Click увеличил продажи книг, но лично я благодаря этому интерфейсу трачу на покупку книг вдвое меньше времени.

Я бессчетное количество раз видел, как это происходит даже с самыми добросовестными и способными программистами. Они берут прототипы экранов, выполненные нами с прецизионной точностью, и рассматривают их в качестве неких предположений в области пользовательского интерфейса. Они берут наши списки функций и возможностей, а затем с легким сердцем выбирают лишь те позиции, с которыми согласны, или те, реализация которых особенно проста.

Общепринятая культура

Сущность войны и потребность в военной муштре одинаковы во всех странах. Отсюда сильное культурное сходство солдат всех стран, не зависящее от того, какую идеологию требуется защитить. То же верно и для компаний, создающих программы.

Коллективная психология *хомо логикус* порождает общую для разработчиков программного обеспечения культуру. Общепринятый способ создания продуктов, основанных на программном обеспечении, поразительно схож для фотокамер и автомобилей, для банков и военно-морских сил. Именно поэтому столь различные продукты, как

фотоаппараты, автомобили Porsche, банкоматы и крейсеры Aegis ведут себя столь одинаковым, узнаваемым, компьютерным образом.

Один из аспектов этой культуры – преклонение перед техническими умениями. Результатом такого преклонения является распространение технических навыков на совершенно иные области, например на проектирование взаимодействия. Тридцать лет назад компьютеры стояли в остекленных зданиях, и работали с компьютерами только программисты. В те времена проектирование, основанное на собственных предпочтениях программистов, было адекватным и уместным. По мере продвижения компьютеров на потребительский рынок программисты продолжали заниматься проектированием по сложившейся традиции. Руководители спрашивают: «Почему я должен платить проектировщикам взаимодействия за работу, которую и так уже делают мои программисты?» Вопрос хороший, если не принимать во внимание ложный тезис, лежащий в его основе. Этот руководитель не получает от своих программистов проектирование взаимодействия – ни бесплатно, ни каким-то иным образом. Скорее, получается интерфейс, радующий только своих авторов – людей с нетипичной подготовкой, нетипичной индивидуальностью и нетипичными склонностями.

Это подчеркивает другой ключевой аспект культуры разработки программного обеспечения. Эта культура, построенная на особенной природе программистов, получает поддержку со стороны руководителей, многие из которых, что скрывать, – бывшие программисты. Джефф Безос говорит, что самый громкий голос в защиту интерфейса 2-Click (в два щелчка) принадлежал менеджеру этого продукта!

Преклонение перед технической квалификацией имеет и другой эффект. Большинство людей считают, что программирование – задача более техническая, чем проектирование взаимодействия. С этим спорить не буду, однако возражаю против заключения, которое обычно делается на основе этого утверждения: что программирование должно предшествовать проектированию взаимодействия в процессе разработки. Ведь в этом случае пользователь вынужден адаптироваться под технологию. Если же проектирование взаимодействия предшествует программированию, технология будет соответствовать целям пользователя. Мне приходилось слышать от руководителей этой индустрии фразы вроде: «Мы включим в процесс проектировщиков после того, как программисты закончат работу над функциональностью». Такой подход ведет к катастрофическому снижению шансов проектировщика взаимодействия внести свою лепту.

Культура программирования в Microsoft

Трудно переоценить роль и значение культуры разработки программного обеспечения. Изучая эту, наиболее типичную компанию, занимающуюся разработкой ПО, Фред Муди (Fred Moody) в своей книге «I Sing the Body Electronic»[22] (Электронное тело пою) показывает, насколько глубоко укоренилась в индустрии программирования культура нердов. Этот писатель и журналист, пишущий на околокомпьютерные темы, провел год в стенах Microsoft, наблюдая за созданием нового мультимедийного продукта, названного в итоге «Explorapedia». Муди получил свободный доступ в Microsoft, и его книга рисует откровенную картину жизни и культуры ведущей компании индустрии. Если вы не поняли этого по продуктам Microsoft, то эта фирма глубоко чтит программирование, но не осознает или почти не осознает потребность в проектировании взаимодействия. Эта книга содержит захватывающее исследование процессов, происходящих в культуре программирования.

Во вступлении Муди готовит почву:

> В Microsoft корпоративная структура формируется из небольших команд, работающих над конкретными продуктами. Команды самостоятельно определяют

[22] Fred Moody «I Sing the Body Electronic», 1995, Viking, New York.

> свою внутреннюю иерархию и сами организуют работу. Это рискованный подход, ведь степень неконтролируемости таких коллективов немыслима в обычных американских корпорациях.

Microsoft известна тем, что нанимает очень одаренных и очень напористых молодых людей чуть ли не со школьной скамьи. Муди пишет:

> «Было такое ощущение, что банда подростков проскользнула в штаб-квартиру какой-то корпорации после окончания рабочего дня и забралась в зал заседаний совета директоров, чтобы поиграть в бизнес».

Microsoft известна и тем, что нещадно эксплуатирует эти молодые таланты, чтобы получить от них как можно больше. Муди пишет:

> «В кампусе всегда очень беспокойно, и там постоянно импровизируют».

Эта книга – замечательный рассказ о том, насколько часто методы разработки в Microsoft произвольны, непрофессиональны и какое морально разлагающее действие они оказывают. Муди и сам был озадачен увиденным, но не сомневался, что увидел нечто важное. А увидел он, что всем здесь заправляют программисты. И даже если не делают этого явно, то делают это опосредованно, силой своей воли. Муди ни в единой букве не подвергает сомнению собственное и общественное мнение, что программистам и *следует* быть у руля, однако отмечает трение, разногласия, неприятные эмоции и ощущение провала, характерные для такой ситуации. Вот что он пишет:

> Не могу сказать, что хорошо понимаю, что происходит в Microsoft. Грустная действительность такова, что покинул я кампус в большем замешательстве, чем когда прибыл туда. Оглядываясь на уже написанные страницы, я прихожу в еще большее недоумение и все еще не в состоянии определить, что за история рассказывается на них – история успеха, или же история поражения, или история успеха, замаскированная под историю поражения, а быть может, история поражения, замаскированная под историю успеха.

Разумеется, Фред стал свидетелем создания танцующего медведя – продукта, безотрадно сложного в применении, единственным достоинством которого стали возможности, не существующие где-либо еще.

Проект Explorapedia – классический пример деградации типичного процесса разработки. Ни минуты не сомневаюсь, что продукт стал провалом. Муди же смутило то, что продукт был сдан вовремя и принес прибыль. На последних страницах книги, озаглавленных автором «Postmortem», он пишет:

> До первого контакта с Microsoft мне и в голову не приходило, что я могу стать летописцем провального проекта. И все же, практически с самого начала и до конца моего пребывания в компании, я считал, что являюсь свидетелем предметного урока в том, как не следует создавать продукт. Все, кто имел отношение к проекту Explorapedia, были настолько несчастны и злы, непрестанно говорили о раздражении и разочаровании, что какой еще вывод я мог сделать, как не тот, что волею судьбы становлюсь свидетелем катастрофы? Однако проект был несомненной победой.

Удивительно? В следующем предложении какие-то два слова отделяют Муди от термина «танцующий медведь», он пишет:

> «Каждая из возможностей Explorapedia в отдельности – лишь бледная

> пародия на изначальный замысел… и, тем не менее, эта энциклопедия стала на рынке единственным продуктом в своем роде».

Легко победить, не имея конкурентов, обладая поддержкой приводящего в трепет брэнда Microsoft, ее связей чудовищных размеров банковского счета.

Слабость продукта – фактор, превосходящий по губительности все прочие. Ближе к концу книги автор цитирует участницу проектирования Сару Фокс (Sara Fox), которая смотрит на …книгу издательства Dorling Kindersley, положенную в основу *Explorapedia.* Она шокирована тем, что книга предоставляет читателям больше свободы в изучении, чем компьютер. Ведь компьютер, предполагалось, станет великой силой, освобождающей от ограничений печатной книги. В книге, указала она, текст свободно окаймлял изображения, и читатели имели возможность листать страницы в свое удовольствие, окидывая взглядом огромные объемы информации. *Explorapedia* же принуждает их рыться в пронумерованных всплывающих окошках, каждое из которых содержит лишь несколько предложений. Это был отвратительный парадокс: компьютер имел ограничений больше, чем книга. «Dorling Kindersley сделало противоположное тому, что делали мы, а мы превратились в посредников».

В Microsoft самые важные проекты задумываются, управляются, реализуются программистами. Проект мультимедийного компакт-диска, описанный в книге Муди, был в некотором роде исключением, потому проектировщики вовлекались в него на каждом шаге. Однако они никоим образом не проявили умения, которые я считаю обязательными для роли проектировщика взаимодействия. Казалось, они совершенно не имеют представления о важных для проектировщика взаимодействия вещах: не понимают до конца, что в действительности делают программисты, не понимают принципы и методы процесса проектирования взаимодействия, не знают состава пользователей продукта и не понимают этих пользователей. Муди ясно показывает, что единственные таланты проектировщиков Microsoft заключались в сообразительности, безграничной энергии и чувстве прекрасного.

Таким образом, Муди и мог увидеть только дисфункциональную модель.

> «Задание проектировщиков состояло в том, чтобы напридумывать как можно больше возможностей, разработчики сопротивлялись во имя сроков сдачи, а работа менеджера продукта заключалась в медитациях и выдаче вердиктов».

Любые антагонистические отношения, подобные описанным, обязательно приводят к тяжелым последствиям. Страдают люди, продукт или же компания.

Сотрудники Microsoft, работавшие над проектом, остались столь же непросвещенными, как Муди. Кевин Геммил (Kevin Gammill), ведущий программист:

> Кэролин постоянно называет это Адским проектом, а Крэйг не перестает повторять, что никогда еще ему не приходилось проходить через подобное. Но Крэйг также не перестает повторять, что мы совершили эту ошибку и еще вот эту ошибку, а еще вон ту ошибку с энциклопедией Encarta, и вот сейчас снова ее совершаем. А Сара твердит, что «жизненный цикл продукта такой… *цикличный».* Здесь каждый проект такой! Мы повторяем, что учимся на своих ошибках… однако снова и снова проходим через ту же [непечатное слово].

Эти неформальные портреты от Геммила захватывают не меньше, чем железнодорожная катастрофа. Читатель, не знакомый с индустрией программного обеспечения, может испытать соблазн списать изложенное на гиперболы или обвинить Муди в выборе нетипичного человека из этой культуры. Однако Геммил – архетип, поэтому его поведение весьма типично. Я встречал сотни мужчин и несколько женщин в точности таких, как он.

Членам той команды было нелегко говорить с Геммилом даже в относительно

нормальной ситуации. Между разработчиками и дизайнерами в Microsoft пролегала гигантская культурная пропасть. Разработчик часто не мог заставить дизайнера понять простейшие элементы проблемы программирования. Так же часто дизайнеры неделями трудились над каким-то аспектом продукта лишь для того, чтобы, показав результат разработчику, получить грубый ответ, что реализация задуманного невозможна.

В последние годы ситуация несколько улучшилась, но эти два лагеря буквально говорили на разных языках, рассматривая мир компьютеров с противоположных интеллектуальных, культурных, психологических и эстетических полюсов. Дизайнеры приходили в Microsoft из гуманитарных дисциплин; разработчики – из мира математики и науки. Разработчики смотрели на дизайнеров сверху вниз, поскольку считали их мышление нечетким и неструктурированным, а вкусы непостоянными. Дизайнерам казалось, что разработчики лишены воображения, консервативны, склонны отвергать предложения по дизайну сразу же, не пытаясь найти способ претворить их в жизнь. Поскольку программирование оставалось для разработчиков необъяснимым таинством, они не имели возможности оценить весомость доводов разработчиков о том, что нарисованное невозможно реализовать.

«Дизайнеры, – любил говаривать Том Корддри, – это непременно женщины, они болтливые, живут в мансардах, сидят на вегетарианских диетах и носят в ушах найденные предметы. Разработчики – обязательно мужчины, питаются приготовленным на скорую руку и произносят только одно слово: „Неверно“.

Он мог бы еще добавить, что дизайнеры и разработчики разрешают конфликты различным образом. Когда разработчики, склонные к вспышкам озорных игр, начинают осыпать дверь кабинета дизайнера шариками из детского помпового ружья, жертва жалуется вышестоящему начальству. Разработчик же сам открывает ответную стрельбу.

Я хочу подчеркнуть, что Microsoft и Муди говорят «дизайнер» там, где я употребляю слово «графический дизайнер». Графический дизайнер обладает развитым чувством прекрасного, мыслит зрительными образами, умеет делать наброски или рисовать. Такой человек участвует в каждом, без исключения, нашем предприятии по проектированию. Однако дизайнеры вносят свою магию в наши работы лишь после того, как подготовленные проектировщики взаимодействия завершат основную работу по концептуальному и поведенческому проектированию.

Кстати сказать, сварливое «неверно», цитируемое Корддри, – хорошая иллюстрация к фразе По Бронсона: «Не я ответил неправильно, вы задали не тот вопрос».

Муди очень хорошо осознавал уникальные культурные отличия программистов и посвятил множество красочных пассажей описаниям их резкого, высокомерного, претенциозного отношения, однако он не смог оценить этих людей. В своем описании контакта программиста Геммила, питающегося гамбургерами, и женщины-«вегетарианки», дизайнера Кэролин Бьерк, Муди дает в корне неверное истолкование:

> Ответы Геммила на вопросы Бьерк больше походили на игривое поддразнивание, однако его манера поведения и поза, несомненно, говорили о враждебном настрое. Он сидел, выпрямив спину, словно проглотил аршин, непрерывно постукивал по полу ногой и барабанил пальцами по столу. Создавалось впечатление, что он предпочел бы находиться в любом другом месте, но только не здесь. Его реакцию на вопросы Бьерк можно было измерять по частоте касаний ногой пола и барабанной дроби пальцев. Частота эта повышалась с ростом сложности реализации возможности, о которой шла речь.

Муди относит раздражение Геммила на счет «сложности» предприятия. Сильнее ошибиться невозможно. Программисты *обожают* сложности. Чем сложнее проблема, тем больше удовольствия в ее решении. Сложность часто становится основным мотивирующим фактором для хороших программистов. Раздражение Геммила происходит из перспективы писать скучный код, и еще – из-за утраты полного контроля в пользу человека, которого он

не уважает, то есть в пользу Бьерк, которая не имеет отношения к техническим вопросам и чьи решения кажутся Геммилу взятыми с потолка. Разумеется, Геммил никогда не скажет об этом открыто, он и сам, вероятно, этого не осознает – но будет использовать сложности, как отвлекающий маневр, чтобы снять с себя ответственность.

Человек, собирающийся возглавить команду разработчиков, должен пользоваться их уважением. Работа программистов устрашающе сложна и предъявляет высокие требования, и программисты ревностно защищают свою территорию. Любой, кто попытается возглавить программистов, потерпит поражение, если только не знает и не уважает работу программистов во всех аспектах. В Microsoft, да и в большинстве других компаний, есть программисты, а есть «мелкие» люди, и эти мелкие люди не смеют даже надеяться повлиять на цикл разработки продукта.

При этом Microsoft имеет несомненный успех, что порождает печальный побочный эффект. Многие компании копируют культуру Microsoft, стремясь повторить ее успех. Копирование атрибутов успеха вместо его причины – ошибка распространенная. Это все равно что увидеть револьверы генерала Джорджа Паттона, украшенные перламутровыми рукоятями, и прийти к ошибочному выводу, что можно стать хорошим стратегом, только если носишь изысканное личное оружие.

Непреднамеренно Муди отмечает еще один интересный аспект нашей культуры разработки программного обеспечения. Многие руководители, обладающие богатым опытом в создании и продвижении на рынках программных продуктов, никогда не применяли проектирование взаимодействия. При этом одни продукты оказывались успешными, другие терпели неудачу, тогда как процесс создания оставался неизменным. Отсюда они сделали вывод, что успех или поражение программного продукта зависит от фортуны; успешная программа – это все равно, что выигрыш на скачках. В истории Муди все говорило о провале, а продукт стал успехом. В случае General Magic, речь, о которой шла в главе 6, все указывало на успех, а продукт провалился. Поиск в ошибочных местах не позволил им обнаружить закономерность, и они просто предположили, что результаты случайны. Ситуация напоминает историю о врачах девятнадцатого века, которые не знали, что является причиной малярии, пока не выяснилось, что переносит заразу анофелес, малярийный комар. Тогда считалось, что заболевание разносится вечерним воздухом и выбирает жертв случайным образом, а единственная защита от этой смертоносной лихорадки – удача. После обнаружения правильной причинно-следственной связи заболевание быстро победили.

Культурная изоляция

В большинстве компаний-разработчиков наиболее опытные программисты берут на себя ответственность за самые сложные части программы. Взамен они получают некий барьер против раздражающих звонков по вопросам технической поддержки: когда звонят реальные конечные пользователи программы, их соединяют с сотрудниками службы технической поддержки или менее заслуженными программистами. В тех редких случаях, когда пользователю удается добраться до ведущего программиста, это происходит потому, что пользователь продемонстрировал свои познания младшему программисту или сотруднику службы поддержки. Следствие такой фильтрации: чем более высоким рангом обладает программист, тем меньше он контактирует с типичными, заурядными пользователями. Он начинает ошибочно предполагать, что «его» пользователи являются типичными.

Пример: Sagent Technology, поставщик систем управления данными для рынка корпоративных вычислений. В этой компании главным специалистом в области баз данных является Влад Горелик (Vlad Gorelik), о компетентности которого в программировании ходят легенды. Непосредственно он общается лишь с теми клиентами, которые способны трепаться о «сегментации запросов», «распределении задач» и «кубах данных» с той же степенью увлеченности. И не удивительно, что Влад считает типичного пользователя Sagent

Information Studio бывалым специалистом по базам данных.

Напротив, Алиса Блэр (Alice Blair), менеджер компании по Information Studio, проводит львиную долю времени в разговорах с потенциальными покупателями продукта. Она объясняет этим людям, что может продукт, каковы его базовые функции. Как следствие, Алиса считает, что в пользовательской аудитории много тех, кто не знаком с продуктом, и тех, кто обладает лишь базовыми навыками работы с компьютером. Неудивительно, что, по ее мнению, большинству клиентов требуется поддержка.

Кендал Косби (Kendall Cosby) работает в службе технической поддержке Sagent. Он не общается с экспертами и новичками. В основном ему приходится работать с конечными пользователями среднего уровня. Поскольку продукт применяется для поддержки принятия решений, Кендал находится в постоянном контакте с финансовыми и рыночными аналитиками, которые мало что знают о компьютерах и базах данных, но в своей работе зависят от возможности обращаться к хранилищам данных и анализировать тенденции в продажах. Собеседники Кендала не очень хорошо разбираются в компьютерах, и ему хотелось бы, чтобы продукт скрывал сложную функциональность или не содержал таковой вовсе. Из всех троих наиболее точным видением клиента обладает Кендал, однако именно Влад и Алиса имеют больше возможностей влиять на архитектуру продукта, поскольку занимают соответствующие должности.

В старинной притче несколько слепых впервые в жизни встречают слона. Первый трогает слона за ногу и объявляет, что это «очень похоже на дерево. Второй трогает слона за бок и объявляет, что это „очень похоже на стену“. Третий трогает слона за хобот и объявляет, что это „очень похоже на змею“. Подобно этим слепым, Алиса, Кендал и Влад имеют весьма различающиеся мнения о том, на что похожи клиенты, поскольку общаются с непересекающимися подмножествами пользователей. Более того, каждый обладает четкими эмпирическими свидетельствами в подтверждение своих выводов. И чтобы получить точный портрет, необходим человек, отвлеченный от ежедневных вопросов, как разработки, так и продаж.

Шкурный интерес

Весьма значимым фактором в культуре разработки программного обеспечения является уединенность. Программисты сидят поодиночке. Конкретный код набирается только одним программистом. Код в компьютере по преимуществу невидим, и его практически никогда не читают. Чтение чужого кода походит не на чтение книги, а скорее на чтение чужих конспектов, записанных непостижимой тайнописью. Программирование настолько сложно, что требует целеустремленного сосредоточения и многих часов непрерывной работы. Программисты чутко относятся к этой замкнутости и всему, что с ней связано. Никто не способен проконтролировать, что делает в своем коде программист. Программисты знают, что качество их кода – вопрос, главным образом, их же собственной добросовестности. Начальство может требовать качества, но не будет тратить время и силы на то, чтобы удостовериться в существовании этого качества. Расшифровка кода может отнять больше времени, чем было изначально затрачено на его написание. Программистам известно, что их личные решения и действия оказывают большее влияние на конечный продукт и удовлетворенность пользователя, чем какие-либо другие соображения. В конечном итоге они лично будут ответственны за успех продукта. Они знают, что глубоко заинтересованы в успехе игры.

Одиночество программиста обостряет его ощущение собственной власти. Некоторые испытывают дискомфорт, но еще больший дискомфорт доставляет им передача полномочий тем, кто не столь заинтересован в исходе. К советам маркетологов, руководителей, проектировщиков программисты относятся со здоровой долей скептицизма. Они знают, что приняв совет, который окажется ошибочным, примут всю вину за произошедшее, потому что к моменту наказания советчика уже и в помине не будет.

Когда программистам разрешается самостоятельно выполнять проектирование взаимодействия, это приводит к некачественному проектированию, а кроме того имеет еще и дополнительный эффект: программисты теряют уважение к процессу проектирования.

Программисты уже так много раз успешно продирались через процесс проектирования, что привыкли игнорировать его ценность. Когда компания, наконец, нанимает проектировщика взаимодействия, программист само собой, относится к его работе снисходительно.

Это приводит к общему недостатку уважения к проектировщику, к процессу проектирования, и, что прискорбно, к собственно проектированию. Такое неуважительное отношение укрепляет внутрикультурную оценку проектирования, как мнения или туманного совета, тогда как на деле это четкое, конкретное и недвусмысленное указание. Поскольку программист, имея на это все основания, считает, что его прихоть имеет вес не меньший, чем чье-то простое мнение, он полагает себя вправе выбирать приглянувшиеся элементы архитектуры из спецификации. Он считает письменную спецификацию не чертежом, но страницей газеты, на которой публикуются письма в редакцию. Некоторые абзацы занимательны, но неверны, другие полезны, но не имеют отношения к делу. К несчастью, программист принимает эти решения, исходя из соображений реализации или собственных предпочтений, а потому решения часто ошибочны.

С другой стороны, каждый программист имеет в запасе ужасные истории о том, как хорошие продукты превращаются в неудачные из-за дурацких указаний, исходящих от руководителей, точно так же не понимающих, чего может хотеть пользователь. Мне вспоминается один высокопоставленный руководящий работник, который ненавидел клавиатуру, а потом требовал, чтобы все программы компании управлялись только мышью. А другой высокопоставленный руководящий работник, неуклюжий в обращении с мышью, заявил, что все программы компании должны управляться только с клавиатуры. Это пагубное проектирование, основанное на личных предпочтениях, вызвало в обеих компаниях взрыв отчаяния.

* * *

Конечно, существуют программисты, сознательно выбирающие злобное и разрушительное поведение, однако, если судить по тем многим программистам, что встречались мне, они столь же редки, как зубы у курицы. Программисты – как пилоты истребителей: после длительного обучения и трудного достижения пика своих способностей они неизбежно начинают считать непрограммистов менее компетентными. Разработчики программного обеспечения уважают других в привычных областях деятельности, но если непрограммист ступает в мир программирования, как описывал Муди, программисты ведут себя покровительственно или даже высокомерно.

Программист имеет все основания презрительно посмеиваться над дилетантами, сующими нос в мир разработки программ. Точно так же, если программист постучится в дверь финансового инспектора и начнет проверять выкладки по возвратам, инспектор сможет с полным правом посмеяться над самонадеянностью и заносчивостью сунувшегося не в свое дело программиста.

Сложность как раз в том, что проектирование взаимодействия и реализация взаимодействия тесно переплетаются в типичном процессе разработки. Руководитель может попросить изменить поведение программы, но не попросит программиста сменить методы разработки. Но именно потому, что поведение и реализация столь тесно связаны, невозможно посягнуть на одно, не создав впечатления покушения и на второе. Это одна из трудностей, наблюдавшихся Муди в Microsoft.

Люди, вовлеченные в создание продуктов, основанных на программном обеспечении, хотят, чтобы уж их-то продукты были просты в применении. Как следствие, они постоянно вторгаются на территорию программистов, у разработчиков же никогда нет лишнего

времени, и такие вторжения могут делать их вспыльчивыми. Многие уединяются и лишь по крайней необходимости общаются с теми участниками команды, которые не занимаются программированием. Тамра Хитершоу-Харт (Tamra Heathershaw-Hart) поведала мне о своих приключениях в роли технического писателя, которому требуется получать информацию от программистов.

> Я обнаружила, что подкуп гораздо эффективнее уговоров. В основном использовала шоколад. Подкуп действовал настолько хорошо, что однажды руководитель группы, пав на колени, публично извинялся передо мной, что забыл уведомить об изменениях в продукте. (Да, свое угощение он все равно получил.) В одной компании пристрастившийся к шоколаду инженер рассказывал мне обо всех изменениях, внесенных его коллегами, чтобы *получить* и их шоколад. До открытия такого способа подкупа я потратила массу времени сверхурочно, пытаясь выяснить, каким образом изменился продукт. Подкуп же позволил сократить сверхурочную работу раза в два.

Эта история занимательна потому, что, обладая хоть каким-то опытом в деле разработки программного обеспечения, мы немедленно признаем, что она правдива. Услышь вы историю про инспектора, вынужденного подкупать шоколадкой клерка, работающего с дебиторскими счетами, чтобы получить информацию по сегодняшним депозитам, вы бы пришли в изумление, негодование и отнеслись бы к рассказу с недоверием.

* * *

Многие руководящие работники привыкли, что подчиненные немедленно реагируют на любое указание или даже мягкий намек, исходящий от начальства. Они воображают, что программисты (технический персонал) не слишком высоко находятся на тотемном столбе власти, и поэтому послушно будут следовать указаниям вышестоящих. С точки же зрения программиста руководящий работник в этой игре ничем не рискует, поэтому повиноваться не хочет. Независимо мыслящий разработчик программного обеспечения не изменит свой код просто потому, что кто-то этого потребовал, независимо от важности персоны попросившего.

Если вы хотите изменить существующий код, необходимо, прежде всего, изменить сознание программиста. Он заинтересован как в сохранении существующего кода, так и в том, чтобы избежать кажущихся ненужными усилий, направленных на изменение кода. Нельзя просто потребовать, а тем более попросить, но следует представить рациональную, обоснованную причину для изменений. Причем представить в терминах, понятных инженеру, и из уст человека, кровно заинтересованного в исходе.

В высшей степени точный и разоблачающий анализ образа мыслей и поведения программистов приводит в своей книге Пол Глен (Paul Glen)[23]. Искренне советую прочитать ее всем, кто хочет глубже изучить программистов и их культуру.

Дефицитный образ мыслей

Проектирование программного обеспечения находится под влиянием фактора, который я называю «дефицитным мышлением». На создание этого фактора работают две силы. Новизна индустрии программного обеспечения широко известна, однако именно эта молодость противодействует самоанализу. Мы слишком заняты ассимиляцией новых технологий, чтобы задумываться о недоразумениях, окружающих более старые. Как следствие, индустрия программного обеспечения переполнена мифами и недоразумениями,

[23] Paul Glen «Leading Geeks: How to Manage and Lead the People Who Deliver Technology», 2003, John Wiley & Sons, New York.

которые никто не ставит под сомнение.

Изумительно, но простой и очевидный факт, что компьютеры сегодня намного мощнее, дешевле и быстрее, чем всего несколько лет назад, не осознан до конца практиками разработки программ. Поэтому большинство приложений не слишком усердны в обслуживании пользователей. Наоборот, приложения встают стеной на защиту центрального процессора из-за ошибочного тезиса, гласящего, что он перегружен работой. В результате программные продукты перегружают работой пользователей. Идеолог проектирования Билл Могридж (Bill Moggridge) так говорит об этом подходе: «Будь добр к микросхемам и жесток к пользователю».

За последнее десятилетие невероятный прогресс в области компьютерной техники сделал обычным явлением мощнейшие настольные компьютеры по доступным ценам. Любой студент и любая домохозяйка могут обладать мощью, которой в 1974 году позавидовал бы центр обработки данных компании General Motors. И при всем том для создания программ сегодня в большинстве случаев применяются инструменты, технологии, методы и умонастроения, основанные на дефицитном мышлении. Разработчики привыкли задаваться вопросом: «Уложимся ли? Будет ли реакция достаточно быстрой? Какую некритичную функциональность мы можем исключить, чтобы сделать программу более эффективной?» Из рассмотрения исключаются вопросы, имеющие большее отношение к делу: «Поймет ли пользователь? Можем ли мы представить информацию в осмысленном виде? Подходит ли этот набор инструкций для целей пользователя? Какая информация является для пользователя первоочередной?»

За некоторыми исключениями, процессоры компьютеров проводят подавляющее большинство времени в бездействии. Да, некоторые процессы требуют интенсивных вычислений, но проистекают совсем не так часто, как нас убеждают создатели аппаратного обеспечения, желающие продавать нам самые новые, и самые замечательные, и самые мощные чудеса электроники. Вряд ли в их интересах, чтобы потребитель знал, что его процессор сильно нагружен лишь на очень коротких дистанциях, а 75-80% времени просто бездействует.

Всего два или три десятилетия тому назад компьютеры были настолько слабыми и настолько дорогими, что любая хорошая идея неминуемо наталкивалась на недостаточную мощность головной машины. Главным вектором развития информатики в те времена стала разработка технологий, снижающих нагрузку на дефицитные вычислительные ресурсы. Такие широко распространенные технологии, как реляционные базы данных, коды ASCII, файловые системы, язык BASIC создавались в основном для того, чтобы снизить нагрузку на компьютер. Программы, написанные в те времена, отдавали приоритет производительности в ущерб другим соображениям, таким как простота применения. Однако уже написанный код неистребим, как сама природа, и многие строки этого старого кода, написанного для старых компьютеров, сегодня работают на современных, невероятно мощных системах.

Обесчеловечивает процесс, а не технология

После выхода в свет фильма Чарли Чаплина «Новые времена» (Modern Times) распространилось мнение, что технология нас обесчеловечивает. Не согласен с таким мнением. Еще до появления технологий тираны, варвары, воины обесчеловечивали своих жертв при помощи кулака и камня. Чтобы сделать человека жестоким, не нужны утонченные инструменты, достаточно взгляда или пинка. Нас делает жестокими не технология. Обесчеловечивают технологи, а точнее говоря – процессы, применяемые технологами для создания обесчеловечивающих продуктов.

Разумеется, чем большим потенциалом обладает технология, тем больший ущерб способны нанести неправильные процессы. И напротив, та же технология при правильном проектировании может стать великим даром человечеству. Высокая технология может пойти в любом направлении, окончательное же воздействие определяют люди, ею управляющие.

Интерактивные системы могут и не быть обесчеловечивающими, но чтобы они не были такими, мы должны перекроить методологию разработки, сделав центром внимания людей, применяющих эти системы. И самое важное изменение для этого процесса – необходимо сначала проектировать интерактивные продукты и только тогда начинать программирование. Следующее по важности изменение состоит в том, чтобы сделать ответственными за проектирование подготовленных проектировщиков взаимодействия. В последующих главах я покажу, чего можно достигнуть, предприняв эти шаги.

Часть IV
Проектирование взаимодействия – выгодный бизнес

Глава 9
Проектирование для удовольствия

Альберт Эйнштейн сказал, что нельзя решить проблему, находясь внутри системы, которая ее породила. Я уделил уже достаточно времени тому, чтобы идентифицировать старый образ мышления и показать, почему он неработоспособен. Теперь настало время поговорить о новом методе, который *работает.* Начиная с 1992 года я занимался разработкой такого метода, получившего название целеориентированного проектирования (Goal-Directed Design), и проектировщики в моей консультационной компании применяют его во всех проектах. В основе метода лежат нетрадиционные подходы к проблемам, ряд мощных руководящих аксиом, а также некоторые поразительно эффективные мыслительные инструменты. В следующих трех главах я представлю обзор трех наиболее мощных инструментов, а также примеры их применения и результаты, которых можно ожидать.

Персонажи

Принципы действия самых мощных инструментов всегда просты, однако применение таких инструментов весьма нетривиально. Это, несомненно, верно и для инструментов проектирования взаимодействия. Наш самый эффективный инструмент исключительно прост: *это точное описание пользователя продукта и его целей.* Сложность здесь в том, чтобы создать и применить такое точное описание.

Наиболее очевидный подход состоит в том, чтобы найти реального пользователя и расспросить *его,* но этот подход неэффективен по ряду причин, и основная из них такова, что жертва определенной проблемы не наделяется автоматически силой, позволяющей эту проблему решить. Реальный пользователь – источник, конечно же, ценный, и мы уделяем большое внимание, но никогда не позволяем напрямую влиять на принимаемое решение.

Работоспособный подход кажется примитивным, но обладает невероятной мощью и эффективен во всех случаях. Мы выдумываем несуществующих пользователей и проектируем для *них.* Мы называем таких несуществующих пользователей персонажами *(personas),* и они представляют собой необходимую базу качественного проектирования взаимодействия.

Персонажи[24] – не реальные люди, но они представляют реальных людей в процессе проектирования. Это *гипотетические архетипы* реальных пользователей. Будучи

[24] Из двух вариантов перевода «персона» и «персонаж» был выбран второй. Краткая справка: «персонаж» (франц. personnage, от лат. persona – личность, лицо) – действующее лицо пьесы (спектакля), сценария (кинофильма), романа и других художественных произведений. Второй термин ближе к функциональной смысловой нагрузке понятия и легче понимается неподготовленными лицами (программистами и руководителями). – *Примеч. науч. ред.*

воображаемыми, они, тем не менее, определяются достаточно жестко и точно. На практике мы не столько «выдумываем» персонажи, сколько *обнаруживаем* их в качестве побочного продукта процесса расследования. Но мы действительно выдумываем их имена и личные сведения.

Персонажи определяются своими целями. Цели же, разумеется, определяются персонажами. Похоже на тавтологию, но это не так. Свойства персонажей выявляются в процессе изучения и анализа так же, как серия тектонических событий выявляется геологами по слоям осадочных пород: присутствие окаменелостей указывает на геологический пласт, а сам геологический пласт указывает на наличие окаменелостей. В следующей главе я много скажу о целях, сейчас же замечу, что мы выявляем их так же, как выявляем персонажи. Мы определяем, какие персонажи имеют отношение к делу, а также их цели в процессе итеративного совершенствования в рамках начального исследования предметной области.

Как правило, мы начинаем с разумного приближения и быстро сосредотачиваемся на правдоподобном наборе персонажей. Данный итеративный процесс похож на итеративный процесс, применяемый разработчиками программного обеспечения при создании продуктов, но имеет одно фундаментальное отличие. Циклическое совершенствование дизайна проекта и его предпосылок происходит быстро и легко, так как мы работаем на бумаге, работаем с текстом. Но циклическое совершенствование реализованного продукта медленнее и сложнее, поскольку здесь необходимо кодирование.

Проектируйте для одного персонажа

Чтобы создать продукт, удовлетворяющий широкую аудиторию пользователей, как подсказывает логика, необходимо возможно больше расширить его функциональность, охватив, таким образом, как можно больше людей. *Это ошибочная логика.* Вы добьетесь гораздо большего успеха, проектируя для единственного человека.

Представьте, что проектируете автомобиль, удовлетворяющий вкусам широких масс. Можно с легкостью выделить по меньшей мере три подгруппы пользователей: мать с малолетним ребенком, плотник, младший руководящий работник. Мамочке нужна безопасная, надежная машина, просторная, с большими дверями, для перевозки детей, собак, пакетов с покупками и т. д. Плотнику Джо нужен крепкий полноприводный пикап, достаточно большой, чтобы в него поместились лестницы, материалы, мешки с цементом и ящики с инструментами. Младший руководящий работник Сет видит себя в машине спортивного типа с мощным двигателем, жесткой подвеской, откидным верхом и – места в машине должно хватать только на двоих.

Решение задачи показано на рисунке. Такой автомобиль сочетает пожелания каждого водителя: минивэн с откидным верхом, пространством для детей и пиломатериалов. Что за дурацкая, невозможная машина! Даже если ее создать, ее никто не купит. Правильное решение: создать минивэн для Мамочки, пикап для Джо, спортивную машину для Сета.

Создать три различных программных продукта проще, чем создать три автомобиля, потому что единственный продукт, как правило, всегда можно настроить таким образом, чтобы получить три различных варианта поведения (заметим, что работу по настройке

нельзя взваливать на пользователя).

Всякий раз, расширяя функциональность, чтобы охватить еще одного пользователя, вы ставите искусственные ограничители в виде лишних возможностей и органов управления программой на пути всех прочих пользователей. Выясняется, что механизмы, приятные одним пользователям, мешают другим получать удовольствие и удовлетворение. Попытка угодить слишком многим может убить продукт, хороший в других отношениях. Однако если спроектировать интерфейс с учетом одного персонажа, ничто не сможет встать между этим персонажем и абсолютным счастьем.

Роберт Лутц (Robert Lutz), руководитель компании Chrysler, говорит, что 80% участников фокус-группы возненавидели новый пикап Dodge Ram. Но после выхода на рынок машина стала бестселлером, потому что остальные 20% в нее *влюбились*. Любовь людей к продукту, пусть даже этих людей не очень много, – вот ключ к успеху.

Чем больше размер мишени, тем меньше вероятность попадания «в яблочко». Если вы желаете достичь уровня удовлетворенности продуктом в 50%, это невозможно сделать, осчастливив 50% широкой аудитории, Единственный способ достичь цели состоит в том, чтобы выявить эти 50% и стремиться осчастливить их на 100%. И это еще не все. Еще большего успеха можно добиться, если выделить 10% рыночной аудитории и направить свои усилия на то, чтобы вызвать у них стопроцентный *восторг.* На первый взгляд – противоречие, однако проектирование для *единственного пользователя* – самый эффективный способ удовлетворить широкую аудиторию.

Чемодан на колесиках и клейкие бумажки

Чемодан на колесиках – хороший пример эффективности проектирования для одного человека. Этот небольшой чемодан со встроенными колесами и убирающейся ручкой произвел целую революцию в области багажа, а проектировался при этом совсем не для широкой аудитории. Изначально чемодан задумывался для экипажей самолетов, то есть для очень узкой аудитории пользователей. Чистота дизайна продукта принесла этой группе пользователей полное удовлетворение. Остальные путешественники вскоре осознали, что такой чемодан решает и их проблемы тоже. Его легко перевозить по переполненным людьми аэропортам, маневрировать между рядами кресел в самолете и завозить на борт.

После успеха чемоданов на колесиках в целевом сегменте продукт выпустили и на другие рынки. Сегодня в продаже есть большие чемоданы на колесах, чемоданы на колесиках «haute couture», бронированные чемоданы на колесиках, детские чемоданы на колесиках. Сегодня уже непросто купить багажный чемодан без встроенных колес и убирающейся ручки.

Вот другой пример. Арт Фрай (Art Fry), инженер отдела клейких материалов компании 3M, решая свои личные, сугубо специфические задачи, создал, можно сказать, самый распространенный и полезный из офисных аксессуаров. Арт Фрай пел в церковном хоре и постоянно сбивался, потому что бумажные закладки выпадали из псалтыря. Не желая портить церковную собственность скотчем, он принялся за поиски лучшего решения. Он вспомнил о клейком материале, над которым работал за несколько лет до описываемых событий. Материал не пошел в производство, поскольку имел низкий коэффициент сцепления. Арт нанес этот неудавшийся материал на небольшие квадратики желтой бумаги и получил удобные закладки. Вот так родились клейкие бумажки Post-It Note компании 3M.

Счастливые пользователи – невероятно эффективное и ценное приобретение. Сужая фокус, можно получить фанатично преданных клиентов целевого рынка. Как видно из главы 5 «Нелояльность клиентов», преданность покупателей становится хорошей поддержкой в трудные времена. Преданные пользователи не просто готовы покорять горы и вброд преодолевать реки, они также представляют собой самый мощный из известных инструментов маркетинга: преданные пользователи лично рекомендуют вас друзьям. Добившись шумихи вокруг своего продукта, вы сможете завоевать и соседние сегменты

рынка.

Гуттаперчевый пользователь

Наша цель состоит в том, чтобы удовлетворить пользователя, но термин «пользователь» является источником трудностей. Из-за своей нечеткости этот термин бесполезен, как циркулярная пила бесполезна для удаления аппендикса. Нам нужен более точный инструмент проектирования.

Те, кто говорит «пользователь», обычно подразумевают эдакое «гуттаперчевое» создание, которому приходится изгибаться, растягиваться и адаптироваться к потребностям момента. Наша же цель состоит в проектировании *программ,* которые будут изгибаться, растягиваться и адаптироваться к потребностям пользователя. Программисты писали и пишут бессчетные множества программ для этого мифологического гуттаперчевого потребителя. Когда программист находит удобным познакомить пользователя с файловой системой Windows для поиска нужной информации, то определяет гуттаперчевого пользователя как пользователя, способного адаптироваться, продвинутого, образованного в области компьютеров; в других случаях, когда программист находит удобным провести пользователя через сложный процесс посредством бестолкового мастера, то определяет гуттаперчевого пользователя, как покладистого наивного новичка. Проектирование для гуттаперчевых пользователей дает разработчику разрешение писать код как угодно, лицемерно презирая «юзера». Реальные пользователи не эластичны.

Программисты создали выразительную систему, описывающую конструирование программного обеспечения. Хорошие программисты не бросаются глупыми обобщениями на тему различных компьютеров и систем. Программист никогда не скажет: «Это будет хорошо работать на компьютере». На каком компьютере? На какой модели? Под управлением какой операционной системы? С какими периферийными устройствами? Точно так же проектировщику никогда не следует расплывчато говорить о программах, будто они «спроектированы для пользователя» или «дружелюбны к пользователю». Такие слова обычно призваны оправдать навязывание собственных интересов.

В нашем процессе проектирования нет места «пользователю продукта» мы говорим о совершенно конкретном человеке – о персонаже.

Персонаж должен быть конкретным

Чем более конкретными мы делаем персонажи, тем более эффективными инструментами проектирования они становятся. Происходит это потому, что по мере конкретизации персонажи теряют эластичность. К примеру, мы не можем просто сказать, что Эмили пользуется офисными приложениями. Мы говорим, что Эмили пишет письма бабушке при помощи WordPerfect версии 5.1. Мы не позволяем Эмили просто поехать на работу. Мы снабжаем ее темно-синей Toyota Camry выпуска 1991 г., с серым пластиковым детским сиденьем и отвратной царапиной на заднем бампере. Мы не позволяем Эмили

просто пойти на работу. Мы назначаем ее клерком по заведению счетов в бежевом отсеке компании Global Airways в Мемфисе, штат Теннесси. Такая характерная детализация – очень мощный инструмент проектирования и коммуникации. Как следствие, все наши персонажи описываются с тщательным вниманием к деталям.

По мере обрастания Эмили конкретными отличительными чертами происходит замечательная вещь: она становится в представлении разработчиков реальным человеком. Ее можно называть по имени, как реального человека. Она приобретает осязаемое воплощение, которое позволяет в процессе проектирования видеть все предположения с ее точки зрения. По мере того, как Эмили утрачивает гуттаперчевость, мы начинаем видеть, каковы ее умения, какова ее мотивация, чего она желает достичь. Вооруженные этим знанием, мы способны изучить ее в свете предметной области программы и понять, является ли она в действительности архетипом пользователя. Проектировщик, обладающий некоторым опытом, обычно способен синтезировать подходящий персонаж с первой попытки.

Одним из самых важных шагов в успешном определении персонажа является выбор имени. *Персонаж без имени просто не имеет смысла.* Никто не сможет представить себе такой персонаж, как конкретного человека.

В стремлении к равенству я не избегаю людей различных рас, полов, национальностей, но стараюсь не выбирать типичных представителей аудитории, поскольку это всех запутает. Шаблонный персонаж более эффективен, если шаблонность придает ему достоверность. Моя цель не в том, чтобы соблюсти политкорректность, но в том, чтобы всех убедить в реальности моих персонажей. Если персонаж – медсестра, то я скорее сделаю ее женщиной, а не мужчиной, и не потому, что медбратьев не бывает, а потому, что подавляющее большинство составляют все-таки медсестры. Если пользователь – компьютерный техник, то персонажем станет «Ник», прыщавый двадцатитрехлетний молодой человек, бывший член школьного клуба Аудио и Видео, а не «Хелен», отлично сложенная красавица ростом 175 сантиметров, посещавшая школу в Беверли Хиллз. Я стремлюсь к правдоподобию, а не к разнообразию.

Чтобы сделать каждый персонаж более реальным для участников проекта, я люблю дополнять имена портретами, прикреплять к каждому персонажу изображение. Как правило, я покупаю изображения в сетевых фотобиблиотеках, но мне случалось работать и с быстрыми набросками. Можете вырезать изображения из журналов, если угодно.

Четко обозначенный, полностью развитый персонаж становится мощным инструментом. До тех пор, пока пользователь не имеет четкого определения, программист может воображать, будто пользователем является он сам. Четко определенный персонаж – это ключ к подавлению склонности разработчика искажать характеристики пользователей. Задолго до написания первой строки кода качественно определенный персонаж становится невероятно эффективным средством для проектирования взаимодействия.

Персонаж должен быть воображаемым

Важно не путать точное определение пользователя с реальным человеком. Реальные люди представляют огромный интерес, как база для исследований, однако для самого процесса проектирования они обычно бесполезны, а часто и пагубны. Здесь уместна аналогия с вином: тонкий букет хорошего вина отлично подойдет к ужину, а необработанные грозди винограда Каберне Совиньон – с крохотными, жесткими ягодами – способны его испортить. Многие ученые, испытывая глубокое почтение к эмпирическим данным, путают реальных пользователей с воображаемыми, более ценными проектировочными персонажами.

Вторая серьезная проблема, связанная с реальными пользователями, состоит в том, что им, как всем настоящим людям, присущи забавные причуды и аномалии поведения, мешающие процессу проектирования. Такие особенности отдельного индивидуума не характерны для группы. Неприятие одним пользователем непосредственного

манипулирования объектами на экране не означает, что его точку зрения разделяют все пользователи или хотя бы большинство пользователей. Обратное также верно. Наш реальный пользователь может замечательно справиться с какой-нибудь проблемой в понимании сложного взаимодействия, тогда как большинство пользователей этого сделать не смогут. Соблазн приписать такую способность всем пользователям лишь потому, что ими обладает реальный человек, очень силен, но этого соблазна следует избегать.

В частности, мы наблюдаем, как поддаются такому соблазну президенты компаний. Один президент, с которым нам довелось работать, ненавидит клавиатурный набор и желает выполнять всю работу без помощи клавиатуры. Он ввел в действие инструкцию, по которой все программы, создаваемые компанией, должны управляться только мышью. Разумно стремиться к управлению программами лишь при помощи мыши, однако неразумно списывать со счетов всех тех пользователей, которым удобнее работать как раз с клавиатурой. Этот президент – не слишком типичный персонаж.

Описание должно быть подробным, а не идеальным

Если говорить о средствах проектирования, то важнее детальность персонажа, а не идеальность ее описания. То есть важнее определить персонаж насколько возможно подробно и конкретно, чем создать абсолютно правильный персонаж. Это удивительная истина, поскольку она противоречит *цели* проектирования взаимодействия, где правильность всегда важнее точности. Конечная цель состоит в том, чтобы получить программу, которая делает то, что нужно, и мы готовы допустить некоторые трудности в работе с системой, чтобы этой цели достичь.

В подвижных узлах механических устройств не должно быть люфта. Так, поршень должен двигаться в цилиндре с минимальным зазором. Любой люфт приведет к быстрому разрушению поршня. В данном случае, длина поршня не имеет такого значения, как притирка. То же верно и для персонажей. Персонаж должен иметь определение достаточно четкое, чтобы сохранять устойчивость под давлением разработки, и это важнее, чем создать правильный персонаж.

К примеру, проектируя чемодан на колесиках, в качестве персонажа мы могли бы взять Герда, командира рейса Боинга-747 из Ванкувера во Франкфурт.

С одной стороны, мы не можем расширением персонажа охватить *всех* пилотов коммерческих рейсов. Скажем, Соня посещает занятия в Университете аэронавтики Эмбри-Риддл и собирается стать профессиональным пилотом после выпуска. Она летает ежедневно, но только на маленьких одномоторных самолетах, и никогда не ночует вне дома. Если говорить о багаже, то Соня – крайний случай. Если определение Герда распространить на Соню, то персонаж из точного превращается в размытый. Начинаются бесконечные и бесцельные дискуссии о том, является ли Соня пилотом авиалиний, и о том, какие требования предъявляет она к своему багажу.

С другой стороны, проектируя чемодан на колесиках, мы можем в качестве прототипа рассматривать Франсин, стюардессу компании на Reno Air. Она трижды в день преодолевает немалые расстояния, раздавая напитки и пакетики с арахисом. Герд и Франсин совершенно разные личности, но их цели и потребности в области багажа эквивалентны.

Программисты живут исключительными ситуациями, и под этим влиянием выбирают персонажи. Они будут спорить, что Соня законно претендует на роль персонажа, поскольку занимает место пилота. В этом разница – программирование определяется краевыми случаями предметной области, а проектирование – центральными. Если есть хоть какое-то сочинение в центральном расположении персонажа, этот персонаж следует исключить из рассмотрения.

В интересах точного определения персонажей необходимо определить средние показатели. Средний пользователь никогда не бывает математически средним. У среднего человека в моем населенном пункте 2,3 ребенка, но ни у одного жителя города не может

быть такого количества детей. Более полезным представителем мог быть стать Сэмюэл, отец двоих детей, или Уэллс, у которого трое детей. Сэмюэл полезен, потому что он личность. Да, гипотетическая, но обладающая точными характеристиками. Родитель, имеющий 2,3 ребенка, не может существовать, как раз из-за этого невозможного среднего показателя.

Усредненные персонажи уничтожают преимущества конкретности точных. Великая сила персонажей именно в их точности и конкретике. Обобщенные данные сводят эту силу на нет.

Персонажи – самый мощный из имеющихся в нашем распоряжении инструментов проектирования. Они являются основой целеориентированного проектирования. Персонажи позволяют нам видеть масштаб и природу проблемы проектирования. Позволяют точно понять и определить цели пользователя и таким образом определить, что должен делать продукт и чего он может совершенно спокойно не делать. Точно определенный персонаж дает нам определенность относительно уровня владения пользователя компьютером, поэтому мы перестаем терзаться загадкой, для кого проектировать: для дилетанта или специалиста.

Изобретенные персонажи уникальны для каждого проекта. Время от времени мы обращаемся к персонажам из прошлых проектов, но требование точности делает практически невозможным существование двух идентичных персонажей.

Реалистичный взгляд на уровень подготовленности

Один из действительно положительных моментов, связанных с персонажами, заключается в том, что они придают дискуссиям об уровне подготовленности пользователей свежесть реализма. Вариации на тему подготовленности пользователей крайне широки, и персонажи позволяют отчетливо это осознать. Широко распространенная модель подготовленности пользователей приведена в главе 2 «Когнитивное сопротивление». Вершину пирамиды составляют «продвинутые пользователи», которые предположительно знают о компьютерах все, если речь не идет о программировании. Центральный фрагмент составляют «компьютерно образованные пользователи», имеющие базовое понимание принципов работы компьютера, но не представляющие себе всех замечательных возможностей. «Неподготовленные пользователи» – это основание пирамиды; считается, что они до безобразия невежественны и неумны.

Вот некоторые примеры персонажей, разрушающие ложные посылки, на которых строится пирамида.

Рупак работает инженером по установке компьютерных сетей в Лос-Анджелесе. Он ежедневно целыми днями возится с компьютерами, он дока в том, как заставить их работать, однако он не понимает, как они работают. Его выживание на данном месте основано на запасе суеверий и практических знаний, способности к механическому заучиванию и бесконечному терпению.

Шэннон работает бухгалтером в оздоровительном центре в Темпе, штат Аризона. Она совершенно не имеет представления о Всемирной паутине, электронной почте, сетях, файловой системе и практически всех остальных аспектах современных компьютеров, но потрясающе обращается с электронными таблицами Microsoft Excel. Она умеет моментально создавать новые финансовые таблицы с графиками и диаграммами.

Декстер – вице-президент отдела развития голливудской компании Steinhammer Video Productions. У Декстера, перемещающегося между павильонами звукозаписи, карманы двубортного пиджака заполнены: там пейджер, два мобильных телефона, наладонный компьютер и беспроводной модем. В области техники он гигант, способный решить любую проблему. Коллеги постоянно звонят ему, просят найти потерянные файлы, но он действительно очень занят, чтобы терять время на подобное обучение. Майкл ждет ответа на третьей линии!

Роберто – представитель по телемаркетингу J.P.Stone, компании, продающей одежду по

почтовым заказам. Он сидит в своем рабочем отсеке, где-то в пригороде Мэдисона, штат Висконсин, на нем гарнитура, а компьютер он использует для обработки поступающих по телефону заказов. Роберто совершенно не разбирается в компьютерах и высоких технологиях, но он уравновешенный добросовестный работник, обладающий замечательной способностью выполнять сложные процедуры. После нескольких дней тренировки он стал одним из самых производительных и эффективных представителей J.P.Stone. Он говорит: «Мне *нравится* мой компьютер!»

Что интересно, ни Рупак, ни Шэннон, ни Декстер, ни Роберто не вписываются в упомянутую пирамиду. Если оставить в стороне угнетающий характер стереотипов, предложенных пирамидой, она совершенно не отражает характер аудитории пользователей. Чрезмерно упрощенные рыночные модели никак не способствуют решению проблем проектирования.

Персонажи закрывают споры о функциях

Как ни удивительно, вторым крайне важным вкладом персонажей является их большая ценность в качестве инструмента общения. Набор персонажей становится системой, обладающей мощным свойством объяснять наши решения в области проектирования. Более того, это как прожектор, высвечивающий для разработчиков, маркетологов, руководителей очевидную правильность наших решений по проектированию.

Жизненно важно, чтобы каждый в команде проектировщиков не только познакомился с набором персонажей, но чтобы все персонажи стали подобны реальным людям, подобны самим участникам команды разработчиков. Программистам свойственен математический подход и они естественным образом не склонны рассматривать отдельных пользователей, предпочитая обобщение. Это переходит и на их отношение к пользователям, которых они всегда представляют в общих, агрегатных или же усредненных категориях. Они предпочитают говорить «пользователь», а не «Джуди», «Крэндал», «Луис», «Эстелла», «Раджив» или «Фрэн».

До введения в обращение персонажей типичный диалог программиста и руководителя, занятых проектированием взаимодействий, выглядит примерно так:

Программист: «Что если пользователь захочет вывести это на печать?

Руководитель: «Не думаю, что печать в первой версии так уж необходима».

Программист: «Кому-то может понадобиться печать».

Руководитель: «Вероятно, да, но не могли бы мы отложить пока эту функцию?»

Руководитель не может победить в этом споре, поскольку в его аргументах нет логики. Аргумент, независимо от его правдивости, выглядит лишь аморфным желанием руководителя сделать что-то иначе, тогда как логике программиста о «возможных событиях» сопротивляться нельзя!

После разработки набора персонажей мы получаем систему, позволяющую точно выразить, кому и что нужно от программы. Однако программистов тяжело убедить, поэтому типичная беседа нашего клиента с программистом на ранних стадиях выглядит так:

Программист: «Что если пользователь захочет вывести это на печать?»

Проектировщик взаимодействия: «Розмари печать не нужна».

Программист: «Кому-то может понадобиться печать».

Проектировщик взаимодействия: «Но мы проектируем для Розмари, а не для *кого-то* ».

На данном этапе особых перемен нет. Программист по-прежнему применяет слово «пользователь» и по-прежнему живет в мире возможных событий. Однако ввод в действие персонажа Розмари – это уже не аморфное, несформированное желание. Напротив, речь идет о конкретном человеке, обладающем известным набором умений и целей. У нас, наконец, имеется убедительный аргумент.

Но поскольку кодом владеют программисты, они могут делать и делают, что захотят, независимо от силы наших аргументов. Ключ к успеху в том, чтобы заставить

программистов поверить в существование и реальность созданных персонажей. Каждый из наших проектировщиков решительно настаивает на выражении всех вопросов, связанных с проектированием, с помощью именованных персонажей. Мы *никогда* не возвращаемся к понятию «пользователь». Через какое-то время такая последовательность приносит плоды, программисты начинают привыкать к персонажам и называть их по именам. Когда программисты начинают называть их по именам по собственной воле, это малозаметное, но переломное событие меняет природу сотрудничества между проектировщиками и разработчиками.

Перелом наступает во всех наших успешных начинаниях, связанных с проектированием. И тогда происходит переключение на высокую передачу. Беседа звучит теперь иначе:

Просветленный программист: «Розмари захочет это напечатать?»

Довольный проектировщик взаимодействия: «Нет. А вот Джейкобу нужна печать квартальных отчетов».

Просветленный программист: «Ну, если печать нужна так редко, сэкономим время и силы, не будем создавать собственный встроенный генератор отчетов, а лицензируем уже существующий».

Довольный руководитель: «Эдак мы сэкономим на разработке две недели!»

Я видел, как после такого перелома наши клиентские компании меняются коренным образом. Раньше они плутали в дебрях бесконечных споров о возможностях и каждые две недели снова решали уже, казалось бы, решенные вопросы. После описанных перемен вопросы проектирования обсуждаются, разрешаются и остаются разрешенными навсегда.

Некоторые наши клиенты заказали футболки с изображениями важных персонажей для каждого из разработчиков. Другие напечатали постеры с персонажами для помещений, где работают программисты. Эти усилия помогают сплотить программистов ради понимания потребителей продукта.

Персонажи нужны проектировщикам и программистам

Нам приходилось работать в компаниях, где программисты не могли себя заставить называть пользователей по именам и не верили в точных персонажей. Они постоянно скатывались обратно к «пользователям», отчего кошмарно страдали продукты.

Я знаю программиста, который просто не понимает механизм действия персонажей. Под давлением аргументов с моей стороны и со стороны моих коллег он признал, что персонажи важны. При этом он упускает из виду главную идею конкретизации, поэтому склонен слово «персонаж» употреблять в качестве синонима слова «пользователь». Он говорит: «Мы должны обеспечить потребности персонажей». Применяя термин, он тем не менее отвергает конкретику, главный действующий ингредиент, из-за чего подход теряет всякую силу.

Один клиент дал нам лишь несколько дней на составление рекомендаций. Мы создали персонаж по имени Эдгар, и большим количеством деталей этот персонаж обрасти не успел. Затем мы вступили в продолжительные дискуссии с клиентом по вопросам, выходящим за исходные рамки проекта. Мы быстро обнаружили, что Эдгар начал размножаться. Различные команды внутри этой компании воспринимали различных Эдгаров, то есть каждая самостоятельно наделяла его теми или иными качествами.

Профессиональные маркетологи мгновенно принимают процесс разработки персонажей, поскольку он очень похож на то, что они делают на этапе определения рынка. Главное различие между персонажами маркетинга и персонажами проектирования в том, что первые создаются исходя из демографии и каналов сбыта, а последние – исключительно на основе пользователей. Это не одни и те же персонажи, хотя служат одной цели. Персонажи маркетинга проливают свет на процесс продажи, тогда как персонажи проектирования проливают свет на процесс разработки.

Продумывая этапы проектирования, мы можем примерять их результаты к персонажам и видеть, насколько хорошо справляемся. Мы начинаем играть роли, действуем от имени и по поручению персонажей. И благодаря конкретным определениям это нетрудно. Примерив на персонаже продукт или задачу, вы сразу можете понять, удастся ли вам его удовлетворить.

Персонаж пользователя, а не покупателя

Одна из распространенных ошибок заключается в проектировании для человека, близкого к продукту, но непосредственным пользователем не являющегося. Многие продукты проектируются для журналиста, пишущего обзор продукта для потребительской прессы. В сфере информационных технологий руководитель, покупающий продукт, часто оказывается не тем, кому придется продукт применять. Проектирование для покупателя – распространенная ошибка в компьютерном бизнесе.

Разумеется, потребности руководителя в ИТ тоже нельзя игнорировать, но в конечном итоге руководителю больше понравится, если продукт сделает довольным *конечного пользователя.* В конце концов, если конечный пользователь доволен и имеет возможность работать продуктивно, это успех для руководителя в ИТ. Нередко наши клиенты игнорируют данный совет и потворствуют этим клевретам технологии. Одарив реальных конечных пользователей продуктом, руководители тонут в потоке жалоб и обнаруживают, что пользователи не желают иметь дело с продуктом, очаровавшим руководителя. Руководитель обращается к создателю приложения и требует, чтобы взаимодействия стали более удобными для конечных пользователей.

Подбор персонажей

Каждый проект получает собственный набор персонажей в количестве от трех до двенадцати. Мы проектируем не для каждого из них, но все персонажи полезны для выражения пользовательской аудитории. Некоторые создаются лишь для того, чтобы было ясно, для кого мы *не* проектируем. Так, в одном из проектов речь шла о системе управления поддержкой клиентов. Мы дали определения трем персонажам, из которых двое были техническими специалистами, работающими в отделе поддержки. Лео Пирс, младший маркетолог продукта, работал на компьютере ежедневно и время от времени сам обращался в службу поддержки. Элисон Хардинг, специалист компании, перемещалась из кабинета в кабинет со своими инструментами в алюминиевом кейсе, разрешая проблемы сотрудников, подобных Лео. Тедван Верен, представитель службы поддержки, целыми днями разговаривал по телефону с людьми, подобными Лео, и сообщал Элисон, какой кабинет следует посетить и что починить.

Наш клиент, компания Remedy Inc., как раз занимался пересмотром флагманского продукта, Action Request System (ARS), и желал сделать его «более простым в применении». Разработав эти три персонажа (и еще ряд других), мы смогли четко выразить действительные цели проекта.

Тед на тот момент был основным потребителем ARS, но не он стал нашим главным персонажем. Мы могли бы сделать программу более простой для Теда, но это означало бы полный провал. Вместо этого мы решили дать Лео прямой доступ к системе поддержки. До этого, нуждаясь в помощи, Лео был вынужден звонить Теду, который уведомлял Элисон. Полный набор персонажей дал нам четкое представление об участниках этой игры. Мы получили возможность сообщить разработчикам системы, что цель будет достигнута лишь в том случае, если Лео, далекий от техники маркетолог, сможет задействовать систему обработки запросов (Action Request System, ARS) со своего компьютера для вызова техпомощи, не прибегая к вмешательству Теда.

Как только мы объяснили положение дел в терминах персонажей, участники команды сразу поняли, что необходимо снять акцент с Теда и сосредоточить внимание на Лео. Тед занимает место так называемого «отрицательного персонажа». Его существование помогает нам понять, для кого мы *не* проектируем.

* * *

Обнаружив такой персонаж, цели которого уникальны, мы идентифицируем его.

Совершенно не обязательно, чтобы персонажи имели непересекающиеся наборы целей, достаточно, чтобы стремления каждого персонажа четко отличались от набора остальных. Цели Рауля, собирающего на конвейере газонокосилки, отличаются от целей Сесили, контролирующей сборку. Сесили стремится улучшить производительность в целом, избежав при этом происшествий. Рауль желает выполнить разумный объем работы, не совершая ошибок. Практические цели этих людей одинаковы, однако мотивы очевидно различаются. Рауль желает стабильности, Сесили желает получить повышение. Их цели достаточно сильно различаются, чтобы появилась необходимость создать два различных персонажа.

Ключевые персонажи

В каждом наборе персонажей есть хотя бы один ключевой персонаж. Эта личность находится в фокусе процесса проектирования. Ключевой персонаж отличает потребность в удовлетворении, недостижимом при помощи интерфейсов, спроектированных для любого другого персонажа. Для ключевого персонажа всегда существует отдельный интерфейс. В примере с Remedy ARS ключевым персонажем был Лео Пирс.

Нахождение одного ключевого персонажа или сразу нескольких – жизненно важный шаг в разработке набора персонажей. По моему опыту, каждый ключевой персонаж требует отдельного уникального интерфейса. Если у нас два ключевых персонажа, то придется в конечном итоге проектировать два интерфейса. Если у нас три ключевых персонажа, придется в конечном итоге проектировать три интерфейса. Если у нас четыре ключевых персонажа, это означает, это у нас возникли трудности.

Если мы обнаружили более трех ключевых персонажей, это означает, что набор проблем предметной области слишком велик, что мы пытаемся слишком много сделать в один прием. Мы создаем персонажи для сужения диапазона конечных пользователей. Отсюда следует, что если число персонажей слишком велико, значит, мы начинаем действовать против исходной цели создания персонажей.

Подбор персонажей – не просто удобное словосочетание, а средство проектирования, как в физическом, так и логическом плане. После просеивания аудитории полезных персонажей оказывается обычно от трех до семи. Мы собираем на одном листе бумаги информацию о них – имена, изображения, описания должностей, цели и, нередко, обрывки сплетен. Этот документ в одну страницу становится неизменной составляющей нашего процесса. Мы распечатываем копии набора персонажей и раздаем их на каждом собрании, независимо от того, присутствует ли клиент. Каждый проектировщик на всех наших мозговых штурмах и собраниях, посвященных детальному проектированию, постоянно держит перед собой эту страницу. Если на собрании присутствуют представители клиента, мы печатаем дополнительные копии и для них. Каждый созданный для клиента документ содержит эту страницу. Наша цель состоит в том, чтобы сделать персонажи неизбежным ингредиентом. Они настолько важны, что мы навязываем их всем и каждому.

Выполнить качественное проектирование и не выразить его в терминах персонажей – не очень мудрое решение. Слишком уж легко скатиться обратно к разговорам о «пользователе» и утратить с таким трудом приобретенный фокус на конкретных архетипах пользователей.

Пример: Sony Trans Com и P@ssport

В 1997 году компания Sony Trans Com предложила нам замечательную проблему из области проектирования. Sony Trans Com – это отделение корпорации Sony, расположенное в Калифорнии и отвечающее за проектирование и производство развлекательных систем для гражданской авиации. Развлекательные системы подобного рода, позволяющие в полете смотреть фильмы, телепередачи, слушать музыку и играть в игры – большой и прибыльный бизнес. Компания Sony Trans Com разработала новое поколение технологии,

предоставляющее пассажирам новые возможности. Самой впечатляющей возможностью новой системы, получившей название P@ssport, стало работоспособное видео по запросу (video-on-demand). Видео по запросу означает, что Патриция на месте 23A может начать смотреть фильм «Когда Гарри встретил Салли» через десять минут после взлета, тогда как Анна, на месте 16C, может начать смотреть тот же фильм на 45 минут позже, причем каждая из них будет иметь возможность приостановить воспроизведение или перемотать фильм, никак не мешая другой.

Система P@ssport подняла планку развлекательных систем в гражданской авиации на высоту, невообразимую для существующих технических решений. В спинку каждого кресла встраивался экран и компьютер с процессором *Pentium* под управлением Windows 95. В носу самолета располагался мощный кластер компьютеров, хранящий огромные объемы оцифрованной информации. Компьютеры на местах соединялись с кластером оптическим кабелем, причем каждые несколько рядов кресел подключались через выделенные концентраторы, что делало систему исключительно быстрой и поразительно мощной.

Sony проработала над системой много месяцев, прежде чем обратилась к нам за помощью в проектировании взаимодействия. Инженеры в своей работе над системой продвигались разумными темпами, а вот проектировщики зашли в тупик. В кресле самолета может оказаться кто угодно, поэтому проектировщики пытались учесть все возможные варианты, от абсолютного новичка в компьютерах до компьютерного спеца. Они понятия не имели, как угодить всем этим клиентам. Мы, кстати, тоже не имели понятия, но у нас были мощные методы проектирования, в частности персонажи, и мы были уверены, что сможем решить проблему.

Традиционное решение

Sony Trans Com успела спроектировать и создать прототип системы P@ssport с традиционным интерфейсом. Интерфейс очень хорошо ложился на внутреннюю структуру программы. То есть отображал реализацию. По существу, он состоял из глубокой иерархии экранов, через которые приходилось пробираться пользователю, принимая решение на каждом экране. Очевидные минусы такого подхода и заставили фирму обратиться ко мне.

Каждый экран представлял дополнительный уровень иерархии, и требовалось пройти шесть экранов, чтобы выбрать фильм.

Классический пример «необоснованного решения». На каждом шаге пользователь делает выбор, последствия которого неизвестны. На первом надо выбрать вид развлечения: музыка (Audio), фильмы (Video), игры (Games), покупки (Shop) и т. д. Если выбрать «Video», то все остальные варианты исчезают, а следующий экран требует выбрать категорию фильма. И так экран за экраном, пока на шестом уровне пользователю не удается, наконец, посмотреть ролик и согласиться или отказаться смотреть весь фильм. Отказавшись, он и должен вернуться на шесть экранов назад, к самому началу, и снова пройти шесть экранов, чтобы добраться до следующего фильма. Уф!

Поскольку P@ssport функционирует на экране, встроенном в спинку кресла, экран всегда находится в пределах досягаемости пользователя, Сразу было ясно, что сенсорный экран будет отличным естественным решением, превосходящим пульт дистанционного управления, который надо держать в руках. Однако заказчик отверг эту идею. Было очевидно, что двенадцать щелчков на каждый вариант – это много, и среднему пользователю понадобится несколько десятков щелчков, чтобы, наконец, выбрать что-то подходящее. Кроме того, сидящий впереди будет в ярости из-за постоянных прикосновений к креслу сзади. Поэтому заказчик отказался от сенсорных экранов и вернулся к пульту управления, связанному с сиденьем коротким шнуром. Классическая иллюстрация к словам По Бронсона о том, что инженеры продолжают чинить то, что не сломано, пока оно не сломается. Компания просто выплеснула ребенка с водой. Инженеры сожалели о таком решении, однако считали его неизбежным из-за того, что разработка продукта была ограничена во времени.

Описанный шестиэкранный интерфейс – классический пример проектирования по модели реализации, точно отражающего внутреннее строение программного обеспечения. Каждый экран с выбором содержал очень мало контекстной или вспомогательной

информации, поэтому пользователь не мог почувствовать себя уверенно, что и сделало навигацию неприемлемо сложной. Каждый раз, погружаясь уровнем ниже, пользователь терял из виду контекст. Выбор пункта «Video» лишал возможности выбрать (или даже видеть) другие пункты, такие как «Games». На каждом шаге программа совершенно никак не отражала общую картину, поэтому пользователю оставалось только теряться в догадках. И он выбирал «Video», не зная, какие фильмы можно посмотреть и сколько их. Он выбирал единственную категорию фильма, не понимая, что все эти категории означают. Что за фильм «Правдивая ложь» – приключенческий, романтический или это комедия? Добравшись, наконец, до названий фильмов, пользователь утрачивает и эту информацию. Хм, «Стирательная резинка» – это же, вроде, какой-то художественный фильм про кроткого школьного учителя?

Даже на стадии прототипа в интерфейсе уже была красивая трехмерная графика, высокохудожественные пиктограммы, а также метафорическая тема карты-глобуса – все атрибуты хорошего, но пустого интерфейса. Мы называем это «раскраской трупа».

Персонажи

Как всегда, наш процесс проектирования начался со стадии тщательного исследования, состоявшего в основном из бесед и проходившего в стенах Sony. Мы выслушали большинство людей, работавших над продуктом включая руководителя проекта, руководителя разработки, пару инженеров, руководителя отдела маркетинга, а также руководителя группы развлекательного наполнения системы. В результате мы получили хорошее представление о том, чего пытается достичь Sony Trans Com, выпуская этот продукт. Кроме того, мы увидели некоторую историческую перспективу развлекательных систем гражданской авиации с точки зрения бизнеса и технологии. Вооружившись этим знанием, мы принялись за исследования в полевых условиях и выслушали массу сотрудников авиалиний, в частности стюардесс и стюардов из нескольких компаний.

В процессе интервьюирования наша команда проектировщиков постоянно изобретала новые персонажи. Выслушав стюардессу, мы каждый раз создавали персонаж и в результате получили около трех десятков. Чем больше мы слушали, тем больше узнавали, и в конечном итоге стали очевидны сходства персонажей. Обнаруживая персонажи с общими целями, мы собирали из них один. Наступил момент, когда персонажей осталось всего десять: четыре пассажира и шесть сотрудников авиалиний. Как можно догадаться, сотрудники авиакомпании имели формальные описания должностей, их служебные обязанности легко было понять и принять во внимание при проектировании. Крепким орешком оказался персонаж пассажира. Каждый из четырех персонажей представлял своеобразный архетип, включающий обширный сегмент пользователей, но невозможно спроектировать интерфейс для четырех персонажей. Требуется найти общий знаменатель. Вот как выглядела четверка финалистов:

Чак Бургермайстер, деловые поездки. Член клуба стотысячников, летает практически еженедельно. Его богатый опыт полетов означает, что он вряд ли станет мириться со сложными, отнимающими время интерфейсами, равно как с интерфейсами для полных чайников.

Этан Скотт, девятилетний мальчик. Путешествует без сопровождения впервые. Этану интересны игры, игры, еще больше игр.

Мари Дюбуа, говорит на двух языках, деловые поездки. Английский для нее второй язык. Ей понравились разделы покупок, а также развлекательных программ.

Клевис Мак-Клауд, старичок, под семьдесят, с причудами. Стареющий, но еще бойкий техасец, немного стесненный начинающимся артритом рук. Этот персонаж – единственный из четырех, который не имеет компьютера и не умеет компьютером пользоваться.

Пассажиры

Экипаж Odissey AIRLINES

Клевис Мак-Клауд
Возраст: 65,
World Odyssey Class

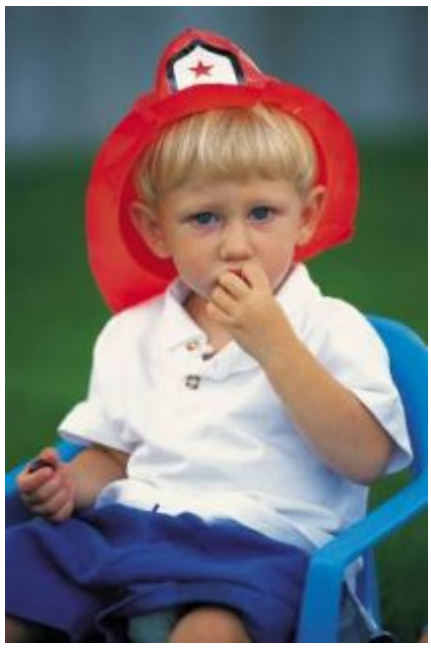
Брент Ковингтон
Возраст: 37,
старший стюард

Мари Дюбуа
Возраст: 31,
Odyssey Club Class

Аманда Кент
Возраст: 28,
стюардесса

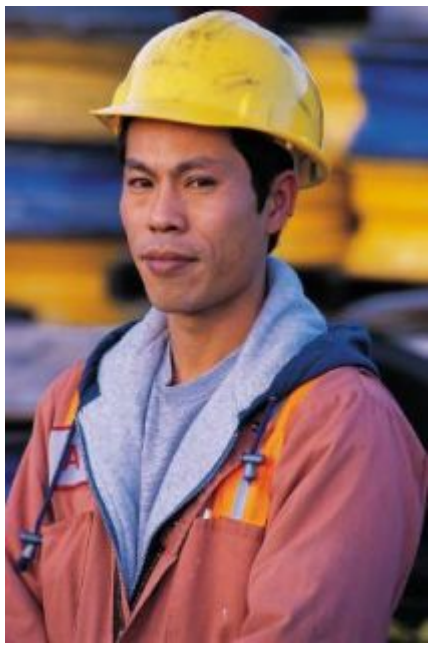
Чак Бургермайстер
Возраст: 54,
Odyssey Gold Class

Жан-Поль Дюро
Возраст: 33,
переводчик-синхронист



Этан Скотт
Возраст: 9,
World Odyssey Class

Молли Спрингер
Возраст: 41,
специалист по обновлению цифровой библиотеки



Мел «Хоппи» Хоппер
Возраст: 51,
механик

Джеймс Таттерсолл
Возраст: 47,
пилот

Наш интерфейс должен удовлетворить Чака, Этана, Мари и Клевиса, не причиняя никому из них *неудобств.* Однако не было речи о том, чтобы сделать всех четверых безумно счастливыми. Этан знает, что игры, игры и снова игры – вещь особая, ради нее можно и несколько лишних кнопочек нажать; лишь бы *был* результат. Чак знает, что обширный опыт полетов позволяет ему экономить усилия на уже привычных вещах, однако, он готов немного напрячься и запомнить специальные команды.

Клевис оказался общим делителем. У него не было компьютера, и он не стремился им

обладать. Его девиз: «Старого пса новым штукам не выучишь», он не глуп и не ленив, а просто не любит высокие технологии. Мы знали, что, поместив на экране диалоговое окно с шапкой и кнопкой «Закрыть», мы моментально потеряем Клевиса. Отсюда следовала полная непригодность привычных компьютерных интерфейсов. Мы также понимали, что его артрит не позволит совершать сложные манипуляции. Система должна подчиниться неточным движениям его рук.

Любое решение, ориентированное на Чака, Мари или Этана, было бы неприемлемым для Клевиса, которого напугали бы и запутали сокращенные команды (Чак) и выбор языка (Мари). Мельтешащие игрушки Этана будут Клевису только мешать; при этом решение, способное осчастливить Клевиса, странноватого старого луддита, будет абсолютно приемлемо для всех остальных, если их особые потребности будут учтены в интерфейсе.

Чак и Марш летают уже давно и смогут сориентироваться в любой системе, если она не потребует длительного обучения. Мы понимали, что простая и наглядная система не подразумевает длительного привыкания, так что Чака и Мари не обидим. С Этаном еще проще, о нем известно заранее, что он быстро и активно освоит систему и разберется со всем, что она предлагает. Он будет доволен, если сможет найти свои игрушки.

На протяжении всего проекта главным ориентиром был Клевис. Его изображение стало нашим боевым штандартом. Идеальная для Клевиса система сделала бы счастливыми и всех остальных пассажиров, всех до единого. Он стал нашим ключевым персонажем, мы проектировали систему для него и только для него.

Проектирование для Клевиса

Клевис не имел опыта работы с компьютерами, как и терпения, позволяющего дождаться вознаграждения, пройдя через многочисленные испытания программы. Решение проблемы навигации для Клевиса было простое: он не может, не будет заниматься «навигацией», поэтому экран может быть только один. Из нежелания Клевиса исследовать интерфейс вытекала необходимость создать продукт, не прячущий информацию. Мы экономим на вариантах выбора, но щедро делимся информацией.

Мы превратили экран в горизонтальную прокручивающуюся ленту, на которой размещались постеры фильмов и обложки альбомов. Мы создали крупную цилиндрическую рукоятку и назвали ее «информационная рукоять». Эту кнопку Клевис мог вращать, как верньер радиоприемника. Рукоятка была не нарисована на экране, а физически присутствовала под ним. При повороте происходила прокрутка ленты с постерами: поворот по часовой стрелке прокручивал ленту вправо, а против часовой – влево.

Клевис просматривает постеры точно так же, как витрины магазинов, идя по улице.

Ему не нужно выбирать категорию фильма и даже задумываться о ней. Мы отказались от древовидной иерархии – никто не будет долбить по экрану, как дятел, поэтому мы вернули к жизни сенсорный экран. Заинтересовавшись фильмом, Клевис касается постера и мгновенно получает рекламный ролик, вместе с текстом, информацией об актерах и стоимости просмотра. Еще одно касание, и он сможет начать просмотр фильма или продолжить прогулку по «Фильм-стрит».

Постеры на ленте расположены в одной плоскости, на единственном слое и распределены по группам. Мы часто заменяем такими группами иерархии в интерфейсах. Предметы на поверхности вашего рабочего стола, вероятно, расположены примерно так же, равно как книги на полках и вещи в ящиках. Все люди, включая Клевиса, Мари, Этана и Чака, очень быстро и легко осваиваются с таким размещением информации. При этом «категория» фильма из *обязательного выбора* превращается в *полезное свойство.* Клевис может присмотреться к постерам и понять, к какой категории принадлежит тот или иной фильм, прежде чем начинать просмотр. Такой подход решает и проблему принадлежности фильмов к категориям. К примеру, фильм «Правдивая ложь» одновременно фигурирует в жанрах боевика, фильма одного актера, приключенческого фильма, фильма с обилием спецэффектов, романтического фильма и комедии – иерархический подход вынудил бы поместить фильм в одну из категорий, а в выбранном нами варианте атрибуты фильма можно занести во все категории.

При прокрутке за постерами фильмов без всякой паузы следуют обложки музыкальных альбомов, а затем постеры игр. Библиотека невелика, и Клевис может быстро добраться до нужного места. Рукоять достаточно крупна, не слишком чувствительна, поэтому даже Клевису с его большими, загрубевшими, уже пораженными артритом руками нефтяника не трудно ее вращать. Панель навигации в нижней части экрана сообщает Клевису, что есть несколько обширных развлекательных категорий, а небольшой индикатор на той же панели перемещается, указывая его текущее положение на шкале выбора.

Программисты Sony попали в ловушку трех чисел – нуля, единицы и бесконечности. На практике система P@ssport способна справиться примерно с тремя дюжинами фильмов. С точки зрения программиста число 36, будучи больше нуля и единицы, эквивалентно бесконечности, а представление бесконечного числа фильмов вызвало осложнения, что и стало причиной возникновения иерархии категорий. Но Клевису по душе прокрутка трех десятков вариантов. Даже если бы фильмов было несколько сотен, ему все равно была бы приятна эта прогулка – он вспоминал бы фильмы, которые уже видел, и предвкушал бы просмотр новых.

Хорошую службу тут сослужили постеры, которые передают существенную информацию о каждом фильме: об актерах, о сюжете, о настрое фильма. Инженеры это понимали, но беспокоились, что присутствие постеров нагрузит поставщиков цифровых фильмов ненужной работой. Когда мы проговорились об идее некоторым из них, реакция

поставщиков была прямо противоположной. Они восторгались при мысли, что можно протащить постеры в интерфейс. В конце концов, они потратили сотни тысяч долларов на создание постеров, передающих наиболее лаконичным способом максимум возможной информации о кинокартине, причем передача эта рассчитана на самую широкую аудиторию. Почему же не использовать результаты этих трудов в самолете? Они сочли это замечательной новой возможностью создать иллюстрации для продукта.

Мы спроектировали интерфейс для одного ключевого персонажа, но предприняли ряд усилий, чтобы удовлетворить нужды и вспомогательных персонажей. Чак Бургермайстер, завсегдатай авиалиний, захочет пользоваться сокращенными командами для быстрого доступа, и такие возможности в интерфейсе присутствуют, незаметные для Клевиса. Если Чаку необходимо переместиться в другую развлекательную категорию быстрее, чем позволяет рукоять, ему достаточно коснуться панели навигации. Программа мгновенно прокручивает ленту до указанной группы, уже без участия Чака. Клевису даже и знать не нужно об этой постоянно доступной возможности, однако ее очень легко обнаружить и изучить, поэтому более опытные путешественники, такие как Чак и Мари, смогут быстро освоиться, самостоятельно или понаблюдав за соседями.

В отличие от изображений на экране, физические органы управления располагают к манипуляциям. Увидев впервые рукоять, Клевис может по ее форме и положению определить, как с ней обращаться. И хотя Клевис не может определить заранее результат вращения, достаточно лишь немного повернуть кнопку, и ее действие становится совершенно очевидным, поскольку экран реагирует прокруткой ленты с постерами. Еще вероятнее, что Клевис увидит, как другие пассажиры вращают рукоятки, а ленты с постерами прокручиваются соответственно. Прямая связь между рукоятью и экраном тривиальна, и вот уже Клевис научился работать с развлекательной системой.

* * *

Я описал лишь интерфейс, спроектированный нами для Клевиса Мак-Клауда, пассажира. Мы спроектировали еще два более емких интерфейса для двух оставшихся ключевых персонажей, Аманды Кент, стюардессы, и Мела Хоппера, механика. Цели этих людей отличаются от целей Клевиса.

Позаботившись о безопасности пассажиров, Аманда должна сосредоточиться на обслуживании, чтобы каждый пассажир остался максимально доволен полетом. Интерфейс для нее должен содержать органы управления всеми операциями в полете. К примеру, если Чак (место 24C) захочет пересесть, потому что Клевис (место 24B) уснул и громко храпит, Аманда должна иметь возможность перенести счет Чака и до половины просмотренный фильм на пустое место 19D, куда он пересаживается.

Основное требование для Хоппи – быстрая оценка состояния системы. Он определяет, какие есть неполадки, насколько они серьезны и что он может сделать, чтобы исправить ситуацию.

Аманда и Хоппи пользуются одним экраном, расположенным на посту стюардов, однако их интерфейсы очень сильно различаются, поскольку различаются их цели.

* * *

При желании проектировать программные продукты, делающие людей счастливыми, вы должны с некоторой степенью уверенности знать, кто эти люди. Вот почему нужны персонажи. Следующий шаг – спроектировать продукт как можно более мощный, а чтобы это сделать, необходимо знать все о целях пользователей.

Глава 10

Проектирование ради результата

Целеориентированное проектирование начинается с создания персонажей и определения их целей. В предыдущей главе я подробно рассказал о персонажах. В этой главе речь пойдет о целях. Я покажу, как определять цели и применять их на практике, в качестве мощного средства проектирования. Персонажи и цели неразделимы, они – как разные стороны одной медали. Персонаж существует, потому что у него есть цели, а цели существуют, чтобы придавать смысл персонажу.

Мы решаем задачи, чтобы достичь целей

До того, как цифровая эра познакомила нас с когнитивным сопротивлением, дизайн (проектирование) был понятием в основном художественным, и мнение одного человека о качестве дизайна продукта было ничем не хуже мнений других людей. Когнитивное сопротивление приходит вместе с взаимодействием, а взаимодействие необходимо лишь в присутствии намерения, цели. В этом новом свете природа дизайна изменилась. Художественная составляющая никоим образом не исчезла. Она лишь попала в тень более серьезной потребности – достижения целей пользователя. Таким образом, в современном проектировании воспринимаемое качество – уже не спорный вопрос, а свойство, которое можно подвергать системному анализу. Иначе говоря, в ярком свете пользовательских целей мы можем достаточно просто определить, какой дизайн будет соответствовать намерениям, независимо от чьего-либо мнения или, если уж об этом зашла речь, эстетических качеств.

Слова «качественное проектирование взаимодействия» обретают смысл лишь в контексте разговора о человеке, непосредственно участвующем во взаимодействиях и имеющем при этом определенные намерения. Намерения не существуют без людей. Эти элементы неразделимы. Именно поэтому ключевыми составляющими нашего процесса проектирования являются цели и персонажи – намерения и люди.

Более того, наиболее важными целями считаются личные цели, интересующие одного конкретного человека. С вашим продуктом взаимодействует реально существующий человек, а вовсе не абстрактная корпорация, поэтому личные цели людей вы обязаны ставить выше целей корпорации. Ваши пользователи будут изо всех сил стараться достигнуть целей бизнеса, но лишь после того, как достигнут собственных. Самая важная личная цель – сохранить достоинство, не почувствовать себя глупо.

Сущность качественного проектирования взаимодействия заключается в изобретении таких взаимодействий, которые помогут пользователям достигать практических целей, не препятствуя достижению личных целей.

Задачи не являются целями

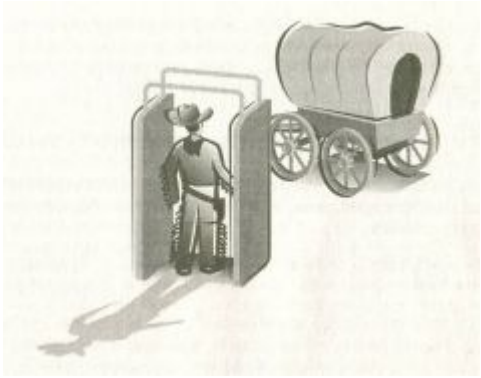
Цели – не то же самое, что задачи. Цель – это конечное состояние, тогда как задача – переходный процесс, необходимый для достижения цели. Очень важно различать задачи и цели, ведь их так легко спутать.

Если моя цель – побездельничать в гамаке, почитывая воскресную газету, то придется мне сначала подстричь лужайку. Моя задача – подстричь газон, тогда как моя цель – отдых. Если бы я мог нанять кого-то для стрижки газона, то достиг бы цели, не прикасаясь к газонокосилке.

Различать задачи и цели просто. Задачи меняются вместе с технологией, тогда как цели обладают приятной особенностью – они очень стабильны. Например, в путешествии из Сент-Луиса в Сан-Франциско мои цели – скорость, удобство, безопасность. Направляясь в Калифорнию на золотые прииски где-нибудь в 1850 году, я путешествовал бы в своем новом, высокотехнологичном фургоне Конестога[25]. В интересах безопасности я взял бы с собой

[25] Конестога (Conestoga) – местность в Пенсильвании, где были созданы фургоны такого типа. – *Прим.*

ружье «винчестер». Направляясь из Сент-Луиса в Сан-Франциско в 1999 году, я путешествую в новом, высокотехнологичном Боинге-777. В интересах безопасности «винчестер» имеет смысл оставить дома. Мои цели остались неизменными, однако задачи изменились вместе с технологиями настолько, что стали прямо противоположными.



Противопоставление целей и задач встречается сплошь и рядом. Если президент желает, чтобы за океаном наступил мир, он посылает пехотинцев, вооруженных автоматами, самолетами, бомбами. Его задача – война. Его цель – мир. Когда адвокат корпорации желает избежать конфликта с коллегой, то вступает в прения о положениях контракта. Цель адвоката согласие, а задача – спор.

Цель – стабильная сущность. Задачи преходящи. Вот одна из причин, по которой проектирование под задачи не всегда уместно, а проектирование под цели уместно всегда.

Программисты занимаются проектированием, ориентированным на задачи

Очень многие разработчики начинают проектирование с вопроса: «Каковы задачи?». Такой подход дает возможность сделать работу, но не позволяет даже приблизиться к наилучшему решению, а также совершенно не удовлетворяет пользователя. Проектирование, ориентированное на задачи вместо целей, – вот одна из основных причин раздражающего и неэффективного взаимодействия. Вопрос «каковы цели пользователя?» позволяет нам смотреть через незамутненное стекло и создавать более качественный и уместный дизайн.

Компьютерное программирование, если добраться до сути, – это создание подробных пошаговых описаний процедур. Процедура есть рецепт решения задачи. Хорошие программисты по необходимости имеют процедурный взгляд на вещи, взгляд, ориентированный на решение задач. В конечном итоге, для достижения целей пользователя задачи необходимо решать, однако существуют различные акценты и различные последовательности выполнения задач. Лишь некоторые из последовательности удовлетворяют личным целям пользователя.

Проектирование, ориентированное на цели

Когда для решения поставленных проблем проектировщики взаимодействия анализируют цели, они обычно находят совсем иные, более подходящие решения.

Цель Дженнифер, офис-менеджера небольшой компании, – сделать так, чтобы дела в офисе шли гладко. Разумеется, она не хочет ни чувствовать себя глупо, ни совершать ошибки. С этой целью она должна обеспечить бесперебойную работу компьютерной сети. Требуется, во-первых, правильно настроить сеть, во-вторых, наблюдать за работой сети и, в-третьих, периодически изменять конфигурацию сети для поддержания максимальной производительности. В представлении Дженнифер ее работа состоит в интеграции всех трех

перев.

задач для достижения *цели* – гладкой работы офиса. С точки зрения Дженнифер эти три задачи действительно не существуют обособленно. Она не видит большой разницы между изначальной настройкой сети и последующей сменой конфигурации.

А теперь обратимся к Клэнси, разработчику программного обеспечения, перед которым стоит задача создать программу для Дженнифер. В присущем Клэнси представлении *хомо логикус* программа решает три задачи (имеет три функции) – под каждую задачу будет отведен отдельный программный модуль. Клэнси кажется естественным, что для каждой из функций отведен собственный участок интерфейса. Ведь это логично. Клэнси думает об интерфейсе, содержащем иерархический список системных компонентов в левой части, а в правой части – информацию о выбранном элементе иерархического списка. Такой интерфейс обладает одним преимуществом – он утвержден компанией Microsoft, а потому кажется программистам разумным. Пользователю придется прощелкать множество системных компонентов, чтобы понять, что происходит с системой, однако вся нужная информация ему доступна только *по специальному запросу*.

На сцену выходит Уэйн, проектировщик взаимодействия, которому поручено осчастливить и Дженнифер и Клэнси. Обладая сознанием проектировщика, Уэйн понимает, что программа должна предстать перед Дженнифер в виде, наиболее точно соответствующем ее целям, и при этом содержать всю необходимую функциональность (здесь Дженнифер – ведущий персонаж). Уэйн также знает, что не может требовать от Клэнси усилий по реализации неразумных или технически невозможных интерфейсных решений.

Уэйну понятно, что у Дженнифер только одна цель – работа без сбоев, и он проектирует интерфейс, позволяющий Дженнифер с одного взгляда увидеть, что все гладко. Если где-то возникает узкое место, интерфейс четко обозначает эту точку сети наглядным способом, так, что она бросается в глаза, и это позволяет Дженнифер исследовать и разрешить проблему, взаимодействуя непосредственно с экранным представлением той области, где эта проблема возникла. Уэйн знает, что для Дженнифер нет разницы между наблюдением за системой и ее настройкой, поэтому интерфейс должен отражать данные ожидания. Ей приходится взаимодействовать с системой в единственном случае – когда она точно знает, что для этого есть причины.

С точки зрения Клэнси, код для отображения производительности компонентов сети и код для настройки этих компонентов – это две различные процедуры. В мышлении, ориентированном на задачи, они не связаны одна с другой. Однако в мышлении, ориентированном на цели, они связаны неразрывно. Дженнифер никогда не займется переконфигурированием, если у нее не будет для этого веской причины, например, может снизиться производительность. Более того, Дженнифер и в процессе переконфигурирования будет внимательно следить за производительностью.

Проектирование для удовлетворения целей персонажа пользователя ясно показывает альтернативный подход к предоставлению функциональности. Часто такой подход дает качественно лучшие способы решения прозаических проблем проектирования. Вот некоторые примеры.

Целеориентированные телевизионные новости

В одном из наших проектов клиент работал над семейством приложений для поддержки процесса создания передачи теленовостей. С точки зрения инженера, мыслящего в терминах задач, такие передачи создаются так же, как строятся мосты – всю передачу делают за один прием. Но мы установили, что у ведущих новостей нет цели «создавать» передачу в течение какого-то времени, у них, скорее, есть цель всегда *иметь в наличии* передачу, которая со временем только улучшается. Каждая передача новостей – это живое существо, очень гибкое и органичное, всю свою жизнь проводящее во взрослом состоянии.

В новостном бизнесе случается всякое, поэтому ведущий всегда желает иметь место для отступления. Его цель – всегда иметь разумную передачу, которую не стыдно выпустить

в эфир. Вечерние новости появляются утром в виде полноценной, готовой к передаче в эфир записи. Эта запись длительностью 22 минуты (не считая рекламных вставок) всегда находится в состоянии полной готовности. Под каждое направление новостей отведено определенное время, и в сумме все сюжеты всегда имеют продолжительность 22 минуты. Здесь можно провести аналогию с подстройкой фокуса размытого изображения: границы новостной передачи никогда не меняются, а вот содержимое становится более точным и выдержанным по мере настройки. Начиная с 10 утра передача готова к выходу в эфир в любой момент, однако лучше всего выйти в эфир приблизительно в 5 часов вечера.

Каждая передача состоит из 20-30 сюжетов, переполненных заставками видеоклипами, репортажами и интервью. В течение утра приоритеты сюжетов изменяются, как изменяется порядок показа и отведенное под сюжеты время, отражая взгляды руководителя передачи. В начале второй половины дня внимания могут потребовать последние известия, изменяя порядок следования других сюжетов и, вероятно, даже исключая некоторые сюжеты из программы. Корреспонденты и режиссер будут вносить исправления и изменения в сценарий до последней секунды, иногда до самого начала эфира.

Разработчики программного обеспечения, рассматривая проблему с точки зрения задач и процедур, создали приложение, позволяющее создавать передачу посюжетно. Что было очень логично, очень разумно, но совершенно неправильно. Передача становилась полноценной только непосредственно перед эфиром, а изменение любого фрагмента разрушало существующую структуру, делая передачу неподходящей для эфира и требуя дополнительной работы над фрагментами.

Мы сделали набросок приложения, работа в котором начиналась с готовой к вещанию в вечерние часы передачи теленовостей. Приложение позволяло корреспондентам и режиссеру постоянно вносить изменения, как если бы вся работа выполнялась вручную. Но помимо этого мы задействовали и мощь компьютерных технологий. К примеру, если сюжет в последнюю минуту удалялся из программы, отведенное под него время автоматически отдавалось оставшимся сюжетам по схеме приоритетного распределения.

Целеориентированное управление классом

В другом проекте нас попросили спроектировать систему управления классами для учителей начальной школы. Разработчики создали модули для проведения тестов, отслеживания успеваемости и доступа к базе данных учебных планов. С точки зрения задач, решение казалось адекватным. Выражаясь метафорически, мы заглянули в глаза учителя, чтобы определить, чего в действительности хочет учитель начальной школы, и получили удивительный ответ.

Мы узнали, что учителя чувствуют себя изолированными в классах, они жаждут информации о том, насколько эффективна их деятельность. Чтобы расти в профессиональном плане, учителю необходим способ, позволяющий оценивать собственную успеваемость. Эта простая потребность неочевидна при разборе процесса обучения на составляющие задачи. В то же время, если исследовать цели, эта человеческая потребность становится очевидна. В своем проекте мы предусмотрели модуль, отслеживающий достижения учителей по семестрам и аудиториям. Этот инструмент позволил учителям получить более осмысленное представление о состоянии дел, что прибавило им уверенности в своей работе.

Цели личные и цели практические

Выше я утверждал, что сущность качественного проектирования взаимодействия состоит в том, чтобы позволить пользователям достигать практических целей, не отказываясь от целей личных. *Хомо логикус* и апологеты обычно находят излишним чересчур пристальное внимание к личным целям и стараются этого избегать. Однако различие между

двумя видами целей – критическая составляющая успеха.

Для примера возьмем моего коллегу Теда. Он только что прислал мне по электронной почте сообщение, в котором жалуется на свой новый телевизор. Много неприятных часов он провел за чтением руководства, потому что иначе не мог настроить многочисленные режимы ящика. Он предположил, что в телевизоре должен быть экранный диалог, сопровождающий пользователя на каждом шаге настройки и который позволил бы обойтись без чтения руководства. Его решение лучше чтения руководства, но, не будучи проектировщиком, Тед, естественно, подошел к проблеме устаревшим, механическим, образом: сосредоточился на задачах. Экранные диалоги упростили бы задачу настройки режимов, но давайте подойдем к проблеме иначе. Мы взглянем на цели Теда, и это даст нам возможность создать решение, качественно превосходящее то, которое предложил он.

Начнем с оценки целей Теда. Здесь всегда предпочтительно начинать с главного. Очевидно, нам известно, что Тед желает смотреть телевизор. Он только что заплатил уйму денег за новый ящик, поэтому так же очевидно, что он желает воспользоваться преимуществами новых супервозможностей этого телевизора. Эти практические цели непосредственно связаны с задачей настройки нового телевизора.

Но мы не должны ни на секунду забывать, что Тед – человек, а значит, обладает выраженными личными предпочтениями, которые можно назвать целями. Тед не хочет, чтобы новая вещь его унижала, он не хочет, чтобы из него делали идиота. Тед не хочет совершать ошибки. Ему нужно чувство достигнутого результата, и чем скорее, тем лучше. Он хочет развлечься. Эти личные цели жизненно важны. С точки зрения проектировщика взаимодействия они важнее, чем практические цели Теда.

Тед жаловался не на то, что не может смотреть новый телевизор, и не на то, что слишком много заплатил, и не на то, что не может воспользоваться новыми супервозможностями. *Он жаловался на то, что телевизор заставил его чувствовать себя глупо.* Конечно, Тед выразился иначе, ведь сама фраза «ящик делает из меня идиота» заставляет человека чувствовать себя глупо, но очевидно, что смысл письма был именно такой. Взаимодействуя с телевизором, он случайно делал ошибки. После подключения телевизора у него ушел час, чтобы получить хоть сколько-нибудь удовлетворительный результат. Процесс настройки режимов вряд ли можно назвать развлекательным.

Взаимодействие, присущее продукту, соответствовало практическим целям Теда, но шло вразрез с его наиболее важными личными целями. Особые качества, сделавшие новый телевизор Теда классическим примером высокотехнологичного танцующего медведя, заключены не в способе достижения практических целей владельца, но в том, как продукт *не смог* удовлетворить его личные цели.

Как бы мы спроектировали новый интерфейс для телевизора, вооруженные информацией о святости личных целей Теда? Во-первых, чтобы владелец быстро почувствовал, что он чего-то достиг, мы должны гарантировать, что телевизор будет хорошо работать сразу после включения. Не нужно, чтобы работали *сразу все возможности,* но *какие-то* должны работать, причем хорошо. Очевидно, этот первый тест на удовольствие невозможно пройти при помощи процесса настройки режимов. Разработчики программного обеспечения относятся ко всем режимам одинаково, а потому валят их в одну кучу. Однако мы можем с легкостью предположить, какими должны быть первичные настройки, что позволит телевизору выполнять базовые функции, а пользователю даст отсрочку в знакомстве с прочими, более сложными возможностями продукта. Необходимо вытаскивать параметры из кучи. Это не техническая задача, а простая перестановка приоритетов в проектировании.

Наш проект телевизора подпадает под определение успешного: Тед может вынуть телевизор из коробки, воткнуть его в розетку и сразу же довольно расслабиться в своем кресле, переключая каналы. Большинство практических целей достигнуто, а личным целям при этом ничто не вредит.

Обратите внимание, что *отсутствие препятствий* в достижении личных целей важнее

мгновенного достижения *всех* практических целей. Это различие также иллюстрирует дополнительные идеи проектирования взаимодействия и обеспечения функциями. Решение должно обеспечить Теду способы достижения всех его практических целей, но проектирование взаимодействия должно четко показать ему, как он может выполнить свои личные задачи.

Принцип соразмерности усилий

Конечно, через какое-то время желание Теда достичь всех практических целей, воспользовавшись преимуществами новых супервозможностей, начнет усиливаться. Но к тому моменту он проведет уже множество счастливых часов со своим новым телевизором, привыкнет к нему и будет готов приложить дополнительные усилия. Теперь уже телевизору будет сложнее унизить Теда, его терпимость к взаимодействиям стала выше, и он стал точнее представлять себе, чего хочет добиться.

Известная человеческая черта, и Тед здесь не исключение, – реагировать на компьютеры эмоционально (более подробно чуть позже в этой главе). Взаимодействуя с продуктом, человек естественным образом наделяет этот продукт человеческими качествами. Тед готов вложить дополнительные усилия в настройку своего телевизора, поскольку чувствует, что телевизор приложил усилия, чтобы доставить ему, Теду, удовольствие.

Это я и называю принципом соразмерности усилий. Люди готовы постараться, решая задачи, потому что воспринимают это, как честный обмен между равными сторонами. Иначе говоря, пользователь готов приложить дополнительные усилия, потому что ожидает получить за эти усилия дополнительное вознаграждение.

Личные цели

Рассмотрим цели более подробно. Я уже говорил о двух видах целей, однако помимо личных и практических существуют еще цели корпоративные и ложные. Личные цели просты, широко распространены и, да, персональны. Парадоксально, но эти качества делают личные цели трудной темой для многих людей, особенно в контексте обезличенного бизнеса.

ЛИЧНЫЕ ЦЕЛИ

Не чувствовать себя глупо
Не совершать ошибок
Выполнить адекватный объем работы
Развлечься (или хотя бы не страдать от скуки)

Апологетов, как правило, очень беспокоит пункт «не чувствовать себя глупо». Это гордые, умные люди, они преуспевают в разрешении сложных ситуаций. Я бы сказал, что это очень похоже на предпринимателей Кремниевой Долины. К примеру, описав историю с телевизором Теда, я в качестве любезности отправил эту историю ему, человеку состоявшемуся, независимому, предпринимателю в области высоких технологий. Вот что ответил Тед:

> Нельзя сказать, что возня с руководством в 40 страниц заставляет меня чувствовать себя глупо. Скорее, хочется сэкономить время, которое я в раздражении потратил бы на решение ненужных задач, а точнее – изучил бы то, что, возможно, впоследствии придется *изучать повторно.* (Скажем, если отключат электричество, потребуется ли программировать телевизор еще раз, снова обращаясь к руководству?)

Тед – апологет. Даже если он всего лишь произнесет резкое слово, это поставит под сомнение его способность победить телевизор независимо от сложности этой задачи, он признает раздражение, потраченное впустую время и даже ненужную избыточность, но ни в коем случае не глупость, даже если это только видимость глупости. Поэтому я неохотно отношусь к идее использовать другое слово. Я применяю слово «глупо» именно потому, что компетентным и разумным, этим суровым высокооплачиваемым гуру Кремниевой Долины так тяжело его произнести. Как они сами говорят, первый шаг к разрешению проблемы – признать, что проблема существует.

Личные цели всегда истинны и действительны в определенных рамках для всех людей. Личные цели всегда предшествуют всем другим целям, хотя очень редко становятся предметом обсуждения – как раз потому, что являются личными. Когда программа заставляет пользователей чувствовать себя глупо, их самооценка снижается, а эффективность деятельности входит в стремительный штопор, независимо от того, какие еще цели существуют. Любая система, идущая вразрез с личными целями, в конечном итоге обречена на неудачу, независимо от того, насколько качественно позволяет достигать целей иного рода.

Корпоративные цели

У каждого делового предприятия свои требования к программному обеспечению и уровень этих требований столь же высок, как и у личных целей отдельного индивидуума. Цель «увеличить прибыли» является преобладающей для совета директоров или держателей акций. Для того, чтобы не отвлекаться от важных вопросов на повседневные задачи и другие ложные цели, проектировщик ориентируется на следующие:

КОРПОРАТИВНЫЕ ЦЕЛИ

Увеличить прибыль
Увеличить рыночную долю
Победить конкурентов
Нанять больше сотрудников
Предложить новые продукты и услуги
Выпустить акции компании в свободное обращение

Психологи, изучающие рабочую обстановку, применяют термин «гигиенические факторы», которые Сол Геллерман (Saul W. Gellerman)[26] определяет как «элементы, обязательные для эффективной мотивации, но не способные мотивировать самостоятельно». Освещение в вашем офисе – гигиенический фактор. Вы ведь не ходите на работу только потому, что там хорошее освещение, но если бы освещения не было, вы не ходили бы туда вовсе.

Я адаптировал этот термин, заменив факторы целями: «гигиенические цели». Я их определяю как цели, необходимые для эффективной работы, но сами по себе не мотивирующие. Все корпоративные и практически цели, приведенные в списке, являются гигиеническими. С точки зрения корпорации, это все важные цели, однако работу выполняет не корпорация, а люди, а для людей важнее цели личные.

Можно провести параллель между целями корпоративными и личными: и те и другие представляют собой высшее выражение целей тех, кому они принадлежат. Ни те, ни другие нельзя умалять. Программа, которая не позволяет достичь какой-либо корпоративной или личной цели, потерпит неудачу.

[26] Saul W. Gellerman «Motivation and Productivity», Amacom, New York, 1963.

Практические цели

Практические цели восполняют пробел между стремлениями компании и стремлениями отдельного пользователя. Корпорация желает, чтобы все работали на пределе с целью максимального увеличения итоговой прибыли. Практическая цель удовлетворения требований клиента соединяет корпоративную цель (более высокие прибыли) с личной целью пользователя (работать продуктивно).

ПРАКТИЧЕСКИЕ ЦЕЛИ

Избегать собраний
Удовлетворять требованиям клиента
Сохранять информацию о заказах клиента
Создавать математические модели бизнеса

Практические цели привлекательнее такой тонкой материи, как цели личные, особенно для бизнесменов и программистов-фанатов. Они создают программное обеспечение, которое замечательно помогает достигать практических целей, но совершенно не способно удовлетворить пользователей. Интерфейс, ориентированный на задачи, провоцирует пользователя на ошибки и снижает их способность быть продуктивными на личном уровне. В результате пользователи чувствуют себя неловко и относятся к программам с подозрением.

Разумеется, чтобы удовлетворить цели бизнеса, вы должны встроить в программу определенные возможности. Пользователь должен решать задачи, необходимые для удовлетворения запросов клиентов и обработки заказов, однако это гигиенические элементы, поскольку любая из возможностей, не связанная с личными целями пользователя, оказывается неподходящей. Если пользователь не может достичь личных целей, то он не способен эффективно достигать целей компании. Непреложный факт: счастливые и довольные сотрудники наиболее эффективны. Еще более справедливой эта истина становится в современной информационной экономике, где действительными активами компании являются люди, а не машины. С другой стороны, если программа игнорирует практические цели и служит *только* целям пользователя, это означает, что вы спроектировали компьютерную игру.

Ложные цели

Повседневные продукты, основанные на программном обеспечении, создаются на основе ложных целей. Многие из этих целей облегчают задачу создания программ, в чем и заключается цель программиста, и потому их приоритет повышается во вред конечному пользователю. Другие ложные цели связаны с задачами, возможностями и инструментами. Все это средства достижения результатов, но еще не результаты, тогда как *цели всегда являются результатами.*

ЛОЖНЫЕ ЦЕЛИ

Экономия памяти
Уменьшение потребности в клавиатурном вводе
Поддержка работы в броузере
Простота в освоении
Обеспечение целостности данных
Ускорение ввода данных
Увеличение скорости исполнения программы

Применение супертехнологии или супервозможностей
Улучшение внешнего вида
Сохранение единообразия интерфейса на различных платформах

«Обеспечение целостности данных» – это не цель для программы управления списками рассылки и, наоборот, цель для программы, вычисляющей орбиты космических челноков. «Экономия памяти» не так уж важна для программы управления пользовательскими базами данных, поскольку объемы данных невелики, а компьютеры мощные. Даже «простота освоения» не всегда оказывается основной целью. Так, пилот истребителя, легко освоивший боевые системы, а затем обнаруживший, что они медленные и неуклюжие, в воздушном бою окажется не в лучшем положении. Цель пилота – победить в бою, а не расслабляться во время учебы.

После изобретения микропроцессора компьютерная революция оседлала волну новой технологии. Любая компания, пренебрегающая новыми техническими идеями, обречена. Однако не следует путать методы и цели. Возможно, *задача* компании, разрабатывающей программы, состоит в применении технологии, но у пользователя *нет* подобных целей. Мне, как пользователю, все равно, как решаются мои рабочие задачи – посредством баз данных иерархических, реляционных, объектно-ориентированных, при помощи структурированных записей в файлах или же просто черной магии. Меня интересует только выполнение работы – быстрое, с некоторой легкостью и достоинством.

Например, в 1996 году фирма Visioneer Company оттяпала изрядную долю рынка настольных сканеров у крепких конкурентов. Visioneer свершила свой замечательный подвиг с помощью старомодных черно-белых сканеров, которые на рынке состязались со сканерами конкурентов, способными сканировать в полутонах и даже в цвете. Однако продукт Visioneer включал программное обеспечение, ориентированное на цели и позволявшее пользователям с легкостью сортировать сканированные изображения и просматривать их, тогда как программы других производителей просто перекидывали изображения в сложную файловую систему.

И у компьютера есть человеческие черты

Клиффорд Насс (Clifford Nass) и Байрон Ривз (Byron Reeves), два профессора Стэнфордского университета, изучают реакцию людей на компьютеры. Путем умелой переориентации классических экспериментов в социальной психологии они смогли пронаблюдать замечательное поведение. Свои открытия Насс и Ривз опубликовали в книге «The Media Equation»[27] (Уравнение media). Авторы убедительно показали, что люди реагируют на компьютеры так же, как на других людей.

Насс и Ривз пишут, что

> «…люди не доросли до уровня технологий двадцатого века»

и что

> «современные средства работы с информацией используют устаревшие подходы… Как следствие, к любой достаточно развитой информационной среде предъявляются такие же требования, как и другому человеку, хотя люди понимают, что это глупо, и склонны впоследствии все отрицать».

Человеческий мозг считает компьютеры не столько неодушевленными предметами, сколько предметами, поведение которых схоже с человеческим, поэтому мы бессознательно

[27] Byron Reevs, Clifford Nass «The Media Equation; How People Treat Computers, Television, and New Media Like Real and Places», Cambrige University Press, 1996.

относимся к ним, как к другим людям, хотя и «считаем, что это неразумно».

Иначе говоря, у людей есть особые инстинкты, подсказывающие, как надо вести себя рядом с другими разумными существами, и как только *произвольный* объект демонстрирует достаточно серьезное когнитивное сопротивление, эти инстинкты включаются, и мы реагируем так, будто имеем дело с другим разумным человеческим существом. Эта реакция бессознательна и неизбежна, она присуща каждому человеку.

Проявив тонкую иронию, Насс и Ривз подвергли испытанию множество студентов, выпускников компьютерных факультетов, причем довольно опытных, чтобы самостоятельно создавать тестовые программы. Подопытные – зрелые, высокообразованные, рациональные личности – все как один отрицали возможность эмоциональной реакции на когнитивное сопротивление, хотя объективные данные свидетельствовали об обратном.

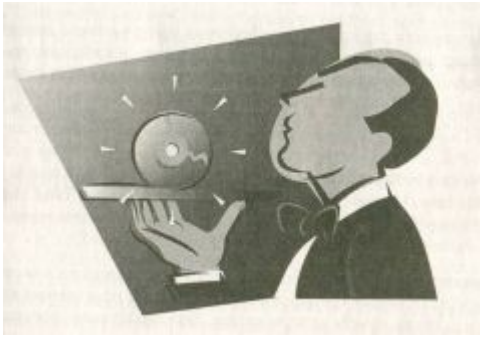
Когнитивист, исследователь мозга Стивен Пинкер из МТИ, подкрепляет этот тезис в своей замечательной книге[28]. Он пишет:

> «Люди продолжают верить во многое из того, что противоречит их опыту, но было правильно в той среде, в которой они развивались. И они преследуют цели, адекватные для той среды, но подрывающие их теперешнее благосостояние».

Проектирование и вежливость

Один важный вывод упомянутого исследования заслуживает особого внимания: если мы хотим, чтобы пользователю понравилась программа, то должны проектировать ее поведение таким образом, чтобы оно напоминало поведение симпатичного нам человека. Если мы хотим, чтобы пользователи эффективно работали с продуктом, то должны спроектировать его так, чтобы он вел себя подобно хорошему сотруднику. Просто, правда?

Насс и Ривз утверждают, что программы должны быть «вежливыми», поскольку это повсеместно распространенная поведенческая особенность человека (набор действий, считающихся вежливыми, для каждой культуры свой, но понятие вежливости существует во всех культурах). Наши продукты, обладающие высоким когнитивным сопротивлением, должны следовать этому простому правилу и вести себя вежливо. Многие высокотехнологичные продукты предполагают, что слова «пожалуйста» и «спасибо» компенсируют любую грубость, но это решительно не то поведение, которое подразумевает вежливость.



Если программа скупа на информацию, затрудняет понимание происходящего, заставляет пользователя долго искать самые востребованные функции да еще торопливо сваливает на пользователя вину за собственные недостатки, пользователю эта программа не понравится, а опыт работы с ней будет неприятным. Наличие слов «пожалуйста» и «спасибо» ничего не меняет. Точно так же ничего не меняет симпатичный, образцовый,

[28] Steven Pinker «How the Mind Works», W.W. Norton & Company, 1997. Мне очень нравится эта великолепная, грамотная, увлекательная книга. Интересное чтение, открывающее глаза.

визуально наглядный, переполненный информацией или даже антропоморфный интерфейс.

С другой стороны, если взаимодействие проникнуто уважением, благородством, содержит полезную информацию, то пользователю программа понравится и работать с ней ему будет приятно. Опять же, этот результат не зависит от типа интерфейса: интерфейс командной строки на зеленом фоне будет принят пользователями, если сможет реализовать перечисленные свойства.

Что такое вежливость?

Что представляет собой дружелюбная к пользователю, или вежливая программа? Каким образом ее поведение может стать более похожим на человеческое? Продавцы подержанных автомобилей одеваются красиво, улыбаются широко, переполнены заманчивой информацией, но становятся ли они от этого приятными? Люди совершают ошибки, медленно думают, импульсивно действуют, но из этого не следует, что программа, обладающая подобными свойствами, хороша. Некоторые люди обладают особыми качествами, и эти качества делают их подходящими для работы в сервисном обслуживании. Программное обеспечение всегда находится в такой роли.[29]

Большинство хороших разработчиков программного обеспечения испытывают затруднения, когда им приходится проектировать вежливость. Роберт Кринджели (Robert X. Cringely) говорит о программистах:

> Они выразительны и точны до чрезвычайности, но лишь когда им этого хочется. Их внешний вид намеренно подчеркивает личные приоритеты, а не свидетельствует о лени. Режим общения у них настолько регламентированный, что иногда, кажется, будто они не способны общаться. Назовите его Майком, если он называет себя Майклом, и почти наверняка он не ответит, поскольку с его точки зрения маловероятно, что вы обратились к нему[30].

Можно видеть, как понятие «вежливость» или даже «человечность» может стать камнем преткновения, попроси мы программистов интерпретировать столь нечеткие концепции. Они сражаются против идеи превращения компьютеров в объекты с более человеческим поведением, потому что считают людей слабыми и несовершенными вычислительными устройствами.

Как-то я спросил своего друга Кейта Плиса (Keith Pleas), а он широко известен в сообществе разработчиков как сложившийся профессиональный программист, чувствительный к вопросам, связанным с пользовательским интерфейсом, о том, как сделать программы более человечными. Кейт интерпретировал «человечность» как внесение неточностей во взаимодействие. Он ответил:

> Что, компьютер будет говорить неправду? Будет сообщать, что на банковском счету осталось «примерно $500»? Будет давать не тот ответ, который минуту назад дал кому-то еще? Если мы добавим человечности, это сделает наши системы менее компьютерными, по крайней мере, относительно текущей ситуации.

Реакция Кейта, как программиста, вполне естественна. Верно, компьютер не сообщает примерные суммы на балансах счетов, а еще компьютер не видит разницы между ответом «примерно $500», полученным за десятую долю секунды, и ответом «ровно $503,47»,

[29] Примечательным исключением из этого правила являются игры. Во многие игры стало бы скучно играть, если бы не скрытые факты, непонятные процессы и нечеткие цели.

[30] Robert X. Cringely «Accidental Empires, How the Boys of Silicon Valley Make Their Millions, Battle Foreign Competition, and Still Can’t Get a Date», Addison-Wesley, 1991.

полученным за семнадцать минут. Действительно вежливая, более человечная программа сразу ответит, что на счету «примерно $500», а затем уведомит пользователя, что через несколько минут сможет сообщить более точную цифру. Теперь *пользователь* решает, тратить ли ему дополнительное время на более точный результат. Это принцип соразмерности усилий: если вы нуждаетесь в большем количестве информации, то подождете, пока она будет получена.

Что делает программы вежливыми?

Люди обладают многими замечательными свойствами, позволяющими им быть «вежливыми», но эти свойства не имеют четких определений. Насс и Ривз пишут, что

> «…четыре базовых принципа, составляющих правила вежливых взаимодействий, это качество, количество, значимость и ясность».

Принципы хорошие, но слишком размытые, чтобы приносить практическую пользу. Вот мой список элементов, повышающих качество взаимодействия, как для людей, так и для высокотехнологичных, основанных на программном обеспечении продуктов, насыщенных когнитивным сопротивлением:

Вежливая программа интересуется мной
Вежливая программа относится ко мне уважительно
Вежливая программа обходительна
Вежливая программа ведет себя разумно
Вежливая программа предвидит мои потребности
Вежливая программа отзывчива
Вежливая программа не склонна делиться своими личными проблемами
Вежливая программа в курсе происходящего
Вежливая программа проницательна
Вежливая программа уверена в себе
Вежливая программа всегда сосредоточенна
Вежливая программа покладиста
Вежливая программа дает мгновенное удовлетворение
Вежливой программе можно доверять

Вежливая программа интересуется мной

Друг интересуется мной, интересуется тем, кто я такой, и тем, что мне нравится. Он запомнит мои предпочтения и антипатии, чтобы в будущем сделать мне приятное. Любой человек, оказывающий услуги, прилагает сознательные усилия, чтобы запомнить лица и имена клиентов. Некоторым нравится, когда к ним обращаются по имени, некоторым – нет, но каждому нравится отношение, учитывающее его личные вкусы.

Программы в массе своей не знают, кто их применяет, да и знать не хотят. Если говорить о моем *персональном* компьютере, то ни одна из *персональных* программ на нем не помнит ни меня, ни фактов обо мне. Факт остается фактом, несмотря на то, что компьютером постоянно, раз за разом, эксклюзивно пользуюсь я и никто иной. Ларри Кили шутит, что писсуар с автоматическим смывом в уборной аэропорта осознает его присутствие в большей степени, чем настольный компьютер.

Каждый фрагмент любой из этих программ должен усердно трудиться, запоминая мои рабочие привычки, и особенно все, что я сообщаю для программиста, пишущего программу; мир представляет собой генератор информации, необходимой в конкретный момент времени, поэтому если программе нужен какой-то клочок информации, она просто требует эту информацию от пользователя. Затем бездумная программа выбрасывает полученные

сведения, предполагая, что при необходимости спросит заново. Компьютер очень хорошо подходит для запоминания информации, так что с его стороны забывать невежливо.

К примеру, в записной книжке моей почтовой программы одиннадцать человек с именем Дэйв. С большинством из них я общаюсь редко, но среди этих людей мой лучший друг Дэйв Карлик (Dave Carlick), которому я постоянно отправляю сообщения по электронной почте. Когда я создаю новое сообщение и набираю в поле адресата неоднозначное имя «Dave», то ожидаю, что программа на основе моего поведения в прошлом сделает вывод, что я имею в виду Дэйва Карлика. Если я захочу отправить сообщение другому Дэйву, Дэвиду Фору, например, то наберу «Dave F», «D4», «David Fore» или что-то подобное, чтобы указать на необычность своего выбора. Но нет, программа каждый раз открывает диалог, заставляя меня выбирать, какого из одиннадцати Дэйвов я имею в виду. Программе на меня наплевать, она считает меня незнакомцем, хотя я – единственный известный ей человек.

Вежливая программа относится ко мне уважительно

Любой работник сферы обслуживания относится уважительно к своим клиентам. Он понимает, что клиент всегда прав, и должен получить то, что желает. Когда в ресторане метрдотель провожает меня к столику, я считаю его выбор столика предложением, а не приказом. Высказав вежливый протест и выбрав другой столик в пустом ресторане, я ожидаю, что мое желание удовлетворят немедленно. Получив отказ, я, вероятно, уйду и выберу другой ресторан, где мои желания ставятся выше желаний метрдотеля.

Невежливая программа контролирует действия человека, которого всегда считает недостаточно компетентным. Приемлемо, если программа высказывает *мнение,* что я допускаю ошибку, однако не приемлемо, если она судит мои действия. Точно так же допустимо, если программа *предполагает,* что я не могу отправить заявку («Submit»), пока не укажу номер социального страхования, но если я все-таки отправлю заявку, не указывая свой номер социального страхования, то ожидаю от программы повиновения. (Само слово «Submit»[31] и обозначаемое этим словом понятие антиподы уважительного отношения. Программа должна подчиняться пользователю, поэтому любая программа, предлагающая кнопку Submit, де-факто невежлива. Обратите на это внимание, держатели активных веб-сайтов.)

Вежливая программа обходительна

Если в аэропорту я спрашиваю у служащего авиакомпании, через какие ворота пойдут пассажиры рейса 79, то ожидаю не только ответа на свой вопрос, но и добровольной передачи крайне полезной дополнительной информации о том, что рейс 79 задерживается на двадцать минут.

Если делаю заказ в ресторане, должно быть очевидно, что мне понадобится также нож, вилка, ложка, стакан воды, соль, горчица и салфетка.

Программы, в массе своей, ничего такого не умеют. Они лишь дают скупые ответы на точно заданные вопросы и обычно не слишком щедры на дополнительную информацию, даже если эта информация однозначно относится к моим целям. Если я пытаюсь напечатать документ, мой текстовый процессор никогда не сообщит, что в лотке осталось мало бумаги или что в очереди перед моим документом еще сорок других, как сообщил бы мне обходительный человек.

[31] В переводе с английского слово «submit» имеет значения «подчиняться», «покоряться», а также «представлять на рассмотрение». В программах и на веб-сайтах речь идет о последнем, однако это значение не является для слова «submit» основным. – *Прим. перев.*

Вежливая программа ведет себя разумно

В каждом хорошем ресторане вам с радостью позволят прогуляться по кухне, при первом вашем визите здравый смысл хозяина подсказывает, что вас надо сопроводить в зал. Похоже, что большинство продуктов, основанных на программном обеспечении, не различают кухню и зал, помещая органы управления востребованных функций рядом с теми, которые никогда не используются. Широко распространены меню со смертельными, необратимыми, катапультирующими функциями, с которыми могут работать только подготовленные профессионалы. Это все равно, что посадить человека за столик рядом с кухонной плитой.

Неподходящие функции в неподходящих местах – вот клеймо продуктов, основанных на программном обеспечении. Отличный пример недостаточно здравого смысла – беспроводной ключ моего автомобиля. Приводившаяся ранее цифра «примерно $500» – хорошая иллюстрация возможности применения здравого смысла в интерфейсе.

Ходит много страшилок о том, как клиенты подвергаются оскорблениям со стороны иррационально рациональных компьютерных систем, постоянно присылающих чеки на сумму $0,00 или на $8943702624,23. Кошмары служб поддержки преимущественно отошли в небытие благодаря продуманной изоляции клиентов от компьютерных систем, однако сотрудникам компаний по-прежнему приходится взаимодействовать с такими системами. Сотрудники получают за это деньги, поэтому не склонны громко жаловаться, да и жаловаться обычно некому – служба поддержки клиентов обычно сотрудникам недоступна.

Вежливая программа предвидит мои потребности

Моя помощница знает, что мне требуется номер в гостинице, когда я еду в другой город на конференцию. Она знает, хотя я не говорю ей об этом. Она также знает, что я предпочитаю комнаты тихие, для некурящих, и бронирует именно такие номера совершенно без моего участия. Она предвидит мои потребности.

Мой веб-броузер проводит большую часть времени в бездействии, пока я просматриваю различные сайты. Он мог бы легко предвидеть мои потребности и готовиться к ним, а не терять зря время. Почему он не может воспользоваться временем простоя для загрузки страниц по видимым ссылкам? Велики шансы, что вскоре я попрошу браузер перейти на одну из этих страниц. Невостребованный запрос легко оборвать, однако на ожидание выполнения нового запроса всегда тратится время. Если бы программа предвидела мои желания, готовилась к моим запросам, вместо того, чтобы молча ожидать моей команды, она стала бы гораздо более отзывчивой без необходимости в более быстром Интернете.

Вежливая программа отзывчива

Обедая в ресторане, я ожидаю, что официант будет реагировать на мои невербальные подсказки уместным образом. Когда я участвую в оживленной беседе с соседями по столику, то ожидаю, что официант займется в этот момент другими делами. Будет совершенно неуместно, если официант прервет наш разговор фразой вроде: «Привет, меня зовут Рауль, и сегодня я ваш официант». С другой стороны, когда наша беседа закончена, я поворачиваю голову и пытаюсь найти глазами Рауля, и в этот момент ожидаю, что он поторопится к нашему столику, чтобы узнать, чего я хочу.

Мой компьютер обычно функционирует в видеорежиме, дающем разрешение 1024x768. На презентациях мне приходится временно уменьшать разрешение до значения 800x600, соответствующего более низкой разрешающей способности видеопроектора. Многие из моих программ, включая Windows 2000, реагируют на уменьшение разрешения изменением размеров и вида окон, а также их положения на экране. Однако я всегда и довольно быстро меняю разрешение компьютера обратно на 1024x768 точек. При этом окна,

автоматически изменившиеся при переходе на более низкое разрешение, не восстанавливают свои настройки, подходящие для более высокого разрешения. Необходимая для этого информация существует в компьютере, но программа просто не восприимчива к моим очевидным потребностям.

Вежливая программа не склонна делиться своими личными проблемами

Мы ожидаем в пивных, салонах красоты, кабинетах психиатров, что бармены, парикмахеры и врачи будут молчать о собственных проблемах и проявят разумный интерес к нашим. Возможно, столь односторонний подход не очень справедлив, но такова уж природа бизнеса услуг. Программы точно так же должны помалкивать о собственных проблемах и высказывать интерес к моим. У компьютера нет своего эго, и для него не существует запретных тем, поэтому он идеально подходит на роль доверенного лица, но ведет себя, как правило, противоположным образом.

Программы постоянно скулят при помощи диалогов подтверждения и хвастают ненужными строками состояний. Я не желаю слышать о том, насколько тяжело трудится компьютер, это излишняя информация. Меня не интересует, что у программы проблемы с уверенностью в себе, когда она не может решить, очищать ли мусорную корзину. Не хочу выслушивать скулеж по поводу того, что она не знает, куда поместить файл на диске. Мне не нужно слышать писк модема или видеть информацию о скорости передачи данных или же об этапах загрузки компьютера, точно так же, как я не желаю получать информацию о разводе бармена, сломавшемся автомобиле парикмахера или же алиментах психоаналитика.

Здесь требуют освещения два момента. Программа не только должна помалкивать о собственных проблемах, но должна также обладать достаточной сообразительностью, уверенностью и компетенцией, чтобы решать такие проблемы самостоятельно.

Вежливая программа в курсе происходящего

С другой стороны, информация о происходящем нужна каждому. Тот же бармен помогает мне, вывешивая прейскурант на видном месте, а также отмечая на меловой доске время начала сбора к субботней утренней игре местных команд, информацию об игроках и ставках.

Владельцам магазинов необходимо информировать покупателей по вопросам, которые покупателям могут быть интересны. Я не хочу, чтобы мясник двадцать первого ноября сообщил мне, что индеек ко Дню Благодарения больше не осталось. Я хочу заранее знать, что запас ограничен, и эту покупку надо сделать пораньше.

При поиске в Интернете с помощью типичной поисковой машины я никогда не уверен в степени актуальности возвращаемых ссылок, что делает поиск бесполезным. Я щелкаю по интересующей меня ссылке и получаю только мерзкое сообщение «404 Link Not Found» (документ по ссылке не найден). Почему бы поисковой машине не проверять периодически каждую ссылку? Если ссылка умерла, то бесполезную запись можно выбросить из индекса базы данных, чтобы я не тратил свое время впустую.

Программы постоянно предлагают мне функции, которые по каким-то причинам в настоящий момент недоступны. Программе следует это понять самостоятельно и не предлагать мне такие функции.

Вежливая программа проницательна

Консьержка нью-йоркского отеля, где я часто останавливаюсь, заметила мой интерес к бродвейским постановкам. Теперь, когда бы я ни приехал, консьержка, причем без моего напоминания, вывешивает в моем номере удобный перечень идущих бродвейских постановок. Она была достаточно восприимчива, чтобы заметить мой интерес, что позволяет

ей предупреждать мои желания и предоставлять мне нужную информацию еще до того, как я успею об этой информации задуматься. Консьержке достаточно несложно воспользоваться своей проницательностью, а я раз за разом останавливаюсь именно в этом отеле.

Работая с приложением, я всегда разворачиваю окно во весь экран. Затем, при помощи панели задач я переключаюсь между окнами приложений.

Однако мои приложения, похоже, не замечают этого, особенно новые. Мне часто приходится разворачивать окна этих приложений, хотя они должны были уже давно заметить мои недвусмысленные и ясные предпочтения. Другие пользователи предпочитают небольшие размеры окон, позволяющие видеть пиктограммы на рабочем столе. Для программы не составило бы труда обнаружить предпочтения пользователя и в дальнейшем действовать адекватно.

Вежливая программа уверена в себе

Я ожидаю от работников сферы услуг уверенности и смелости. Если они видят, как я выхожу из уборной с расстегнутой ширинкой, мне хотелось бы, чтобы кто-то быстро, ясно и ненавязчиво дал мне это понять, пока я не вышел на трибуну, чтобы произнести речь. Тут нужна смелость, но такая смелость оценивается по достоинству. Точно так же, если мой помощник не может забронировать мне билет на нужный рейс, я ожидаю, что он уверенно закажет билет на другой подходящий рейс, и мне не придется вдаваться в детали.

Если я велю компьютеру уничтожить файл, я не хочу, чтобы он спрашивал у меня, уверен ли я. Разумеется, уверен, иначе бы я не просил это сделать. Я хочу, чтобы компьютер следовал своим убеждениям и просто удалил файл.

С другой стороны, если компьютер подозревает, что я мог ошибаться (а он это делает всегда), то должен предусмотреть ситуацию, когда я передумаю, и файл нужно будет полностью восстановить. В любом случае продукт должен быть уверен в собственных действиях, а не отмежевываться и не скулить, перекладывая ответственность на меня.

Мне часто случалось подолгу работать с документом, затем нажимать кнопку Print и уходить за чашкой кофе, предоставив принтер самому себе. Затем я возвращался и обнаруживал бессмысленный ужасный диалог в центре экрана с вопросом: «Вы уверены, что хотите печатать?» Подобные сомнения приводят в ярость и являются антитезой, полной противоположностью вежливого человеческого поведения.

Вежливая программа всегда сосредоточена

Если я заказываю салат в хорошем ресторане, мне приносят хороший салат. В плохом ресторане учиняется допрос: «Со шпинатом, Цезарь-салат или овощной? С луком? С гренками? С тертым сыром? Пармезан или романо? Полная порция или к обеду? Соус французский, итальянский, масло с уксусом, Тысяча Островов? Соус на тарелке? Подавать до или после главного блюда?» Даже самый требовательный гурман не настолько озабочен салатом, чтобы подвергаться такой мариновке, однако интерактивные системы ведут себя подобным образом постоянно. Приложение Adobe Photoshop печально известно своей способностью забрасывать пользователя многочисленными отвратительными и ненужными вопросиками, причем каждый появляется в отдельном диалоговом окне.

Невежливые программы задают множество надоедающих вопросов. Потребность в выборе обычно не так велика, поэтому выбор становится не преимуществом, а сущей пыткой.

Выбор ведь тоже можно предлагать различными способами. Можно расставлять варианты, словно на витрине. Мы разглядываем витрину в свое удовольствие, изучая, выбирая или не обращая внимания на выставленные товары. Другой способ: варианты вываливаются на нас, словно недружелюбный допрос таможенником при пересечении границы: *«У вас есть, что декларировать?».* Таможеннику известно, что можно закрыть

глаза на что угодно, однако раскрытие контрабанды может иметь более чем просто неприятные последствия. Мы не знаем о последствиях вопроса. Будут ли нас обыскивать? Если мы знаем, что обыска не избежать, то не станем лгать. Если знаем, что обыска не будет, то испытаем соблазн протащить через таможню лишний блок Marlboro.

Вежливая программа покладиста

Когда системы ручной обработки информации преобразуются в компьютеризованные системы, практически всегда преобразование происходит с потерями. Компьютеризация ручных систем обычно необходима для повышения емкости, а не для изменения функциональности. Однако системы ручного труда обычно очень гибки, а этой функции непросто дать точное определение. Автоматическая система ввода заказов способна обработать на много миллионов больше заказов, чем человек, но клерк обладает способностью *управлять* системой.

После автоматизации его способность *управлять* системой исчезает. Практически не бывает автоматизированных систем, позволяющих внести элемент хаоса в процесс или воспользоваться каким-то преимуществом.

В ручной системе, если по телефону звонит приятель клерка из службы продаж и сообщает, что ускоренная обработка вот этого заказа означает дополнительные заказы в будущем, клерк может ускорить обработку. Когда поступает заказ с недостающей важной информацией, клерк может обработать этот заказ, сделав мысленную пометку позже добыть необходимые сведения. Компьютеризованные системы такой гибкостью, как правило, не обладают.

В компьютеризованной системе существует всего два состояния: отсутствие информации и полное соответствие информации формату. Промежуточные состояния не распознаются и не принимаются. В любой системе ручного труда существует важное, пусть и парадоксальное, состояние – не озвученное, не документированное, но фундаментальное – это состояние *приостановки,* когда транзакция может быть принята, даже не будучи полностью обработанной. Оператор-человек создает это состояние в своей голове, или на рабочем столе, или в кармане.

К примеру, цифровая система, прежде чем выдать накладную, требует предоставить информацию о покупателе и информацию о заказе. Клерк может просто поместить заказ на обработку, еще не имея полной информации о покупателе, а вот компьютеризованная система отклонит транзакцию, не желая продолжать работу без необходимых сведений.

Эту человеческую способность выполнять действия непоследовательно или до удовлетворения предварительных условий я называю «покладистостью». Покладистость обычно становится одной из первых жертв компьютеризации, а ее отсутствие – главная причина нечеловечности цифровых систем. Это естественный результат применения модели реализации. Программисты не видят причин для создания промежуточных состояний, поскольку компьютер в них не нуждается. Однако человек нуждается в возможности слегка изменять систему.

Большим преимуществом покладистой системы является сокращение числа ошибок. Разрешая присутствие в системе небольших временных ошибок и доверяя человеку их исправление до того, как они приведут к серьезным проблемам, мы избегаем ошибок более серьезных, имеющих далеко идущие последствия. Парадоксально, но жесткие правила компьютерных систем как раз и создаются для предотвращения таких мелких ошибок. Такие негибкие правила делают человека и программу врагами, поскольку человек не имеет возможности изменить работу программы и предотвратить серьезные ошибки, а потому вскоре перестает заботиться о защите программы от действительно колоссальных проблем. Если жесткие правила ограничивают гибких людей, проигрывают обе стороны. Ограничение деятельности людей всегда заканчивается плохо для бизнеса, а компьютерным системам в конечном итоге все равно приходится переваривать неверные сведения.

* * *

Покладистость – одно из немногих свойств вежливости человека, которое сложно встроить в компьютерную систему. Покладистость требует гораздо более тонких интерфейсов. Чтобы стать покладистой, система должна приоткрыть свои механизмы умеренно опытному наблюдателю. Клерк не может переместить форму заказа на вершину стопки, если не может видеть саму стопку, ее размер, границы, расположение. Необходимы инструменты, чтобы вытащить форму из электронной стопки и поместить ее на вершину. Эти инструменты должны быть видимыми, как в системе ручного труда, и тогда операция становится столь же простой, как перемещение листка бумаги. С физической точки зрения покладистость требует дополнительных механизмов для хранения записей, находящихся в состоянии ожидания, но ведь и функция отката операций имеет очень похожие потребности. Настоящая неприятность состоит в том, что такие инструменты допускают мошенничество и злоупотребления.

Искажение работы системы можно интерпретировать, как мошенничество. Технически оно и есть нарушение правил. В мире ручной обработки искажение подразумевается, на него не обращают внимания. Это очень кратковременный, особый случай, и предполагается, что инициатор закончит работу с такими счетами до того, как уйти домой, в отпуск или же на другую работу. Конечно, все подобные случаи тщательно подчищаются перед визитом аудиторов. Если бы процессы легкого нарушения правил были хорошо известны, то это могло бы спровоцировать людей на мошенничество.

А если метод еще и подробно документирован в операционном руководстве компании, что придает ему дополнительный вес, то некоторые слабохарактерные личности могут усмотреть в методе возможность работать неаккуратно и недоделывать работу или же обманом выманить деньги у компании. Поддержка покладистости со стороны компании – шаг финансово безответственный.

Однако покладистость оказывает мощное воздействие на отношение пользователей к системе. Все доводы против покладистости системы очень рациональны и подтверждаются логическими аргументами (вероятно, и с точки зрения закона тоже). К сожалению, все эти доводы описывают идеализированное положение, не давая точного описания реальной рабочей обстановки. Во всех сферах бизнеса все участники пользуются покладистыми системами ручного труда, чтобы поддерживать плавный ход колеса бизнеса, колеса жизни. Очень важно, чтобы автоматизированные системы также пропитывались этим качеством, несмотря на существующие преграды.

Положительное качество недокументированного использования, перевешивающее недостатки, заключается в том, что компьютер обладает мощью для перепроверки всех действий пользователя, способен подробно регистрировать их для внешнего наблюдателя. Принцип здесь очень простой: разрешайте пользователю делать, что ему угодно, но храните очень подробные записи об этих действиях, чтобы можно было с легкостью восстановить ход событий.

Вежливая программа дает мгновенное удовлетворение

Компьютерное программирование – это сказка про белого бычка. Компьютеры не способны ровным счетом ни на что, пока вы не приложите гигантские усилия для написания программы. Разработчики программного обеспечения медленно усваивают этот принцип отложенного удовлетворения и потому склонны писать программы, ведущие себя таким же образом. Программы заставляют пользователей вводить все возможные сведения, прежде чем сделать даже самый крошечный объем работы. Веди себя подобным образом какой-либо человек, он бы вам очень не нравился.

Мы можем сделать программы намного более вежливыми, предположив, что они

работают на пользователя и предоставляют ему информацию, не требуя единомоментно значительных усилий. Возьмите хоть телевизор Теда – Тед должен иметь возможность смотреть программы без необходимости настраивать режимы.

Вежливой программе можно доверять

Между друзьями устанавливается доверие благодаря взаимозависимости и готовности жертвовать собой. О каком доверии может идти речь, когда компьютеры ведут себя странно и с неохотой работают на пользователей? Я доверяю кассирше в банке – ведь она улыбается и знает мое имя. Но при этом всегда пересчитываю деньги, полученные от банкомата, просто потому, что не доверяю этой тупой машине.

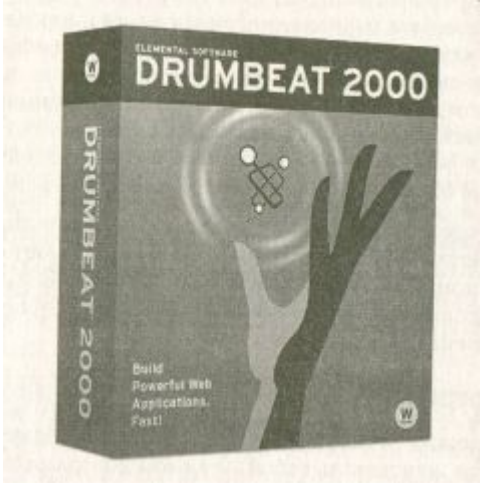
* * *

Программные продукты раздражают нас не отсутствием возможностей, но своей невежливостью. Как показывает этот перечень характеристик, вежливую программу обычно создать не сложнее, чем невежливую. Просто кто-то должен представить себе взаимодействия, эмулирующие качества восприимчивого и заботливого друга. Ни одна из характеристик не противоречит всем прочим, более прагматическим целям вычислительной техники на службе бизнеса. Более человечное поведение может стать и самым прагматичным.

Пример: Drumbeat от Elemental

В числе интересных для нас проектов можно упомянуть небольшую начинающую компанию из Сан-Диего, компанию Elemental Software. Продукт компании, названный Drumbeat, представляет собой инструмент для создания динамических веб-сайтов, основанных на базе данных.

Набор персонажей, разработанный нами для Elemental, оказался незаменимым, хотя и состоял всего из двух очень простых персонажей, не имевших даже фамилий[32]. Создав их и поняв их цели, мы получили понимание, полностью изменившее философию проектирования этого продукта.



Elemental с самого начала захотела многого. Компания намеревалась создать программу, существенно превосходящую по мощности все конкурирующие приложения. Кроме того, программа и в простоте применения должна была превосходить все прочие. Эти

[32] Вообще-то, в полном наборе персонажей было больше, но Бетси и Эрни затмили всех.

цели вполне совместимы. Большинство наших проблем можно отнести на счет приобретения компанией Elemental существующего продукта, на основе которого нам и пришлось работать. Наши желания постоянно вступали в конфликт с уже существующим кодом.

Уже созданный продукт обладал мощными возможностями, но создавался на основе размытых воззрений на конечных пользователей. Имеющимися функциями было трудно воспользоваться, и поэтому продукт получился не слишком мощным. Эд Форман (Ed Forman), новый вице-президент по разработке, сделал ставку на Cooper Interaction Design. Он сам был человеком достаточно новым и еще не до конца завоевал доверие программистов, поэтому наше присутствие могло стать источником переворота. Однако Эд отлично выступил в роли защитника и дал нам достаточно времени, чтобы пообщаться с его командой, узнать их, познакомить их с нашими методами.

Расследование

В своем расследовании мы опросили несколько человек, в основном веб-мастеров. По мере продвижения у нас стала складываться ясная картина. Область разработки для Всемирной паутины удобно делилась на два лагеря. Разумеется, мы создали типичный персонаж для каждого лагеря, и эти два персонажа стали ключом к головоломке Drumbeat, хотя и не совсем так, как мы ожидали.

Через несколько дней после начала проекта мы смогли дать имена и приближенные описания двум веб-разработчикам – Бетси и Эрни.

Бетси – дизайнер. Она носит черное и пьет эспрессо. Раньше была художницей, но потом ее укусил паук Web, и теперь она создает макеты экранов вместо макетов страниц. Она прочла достаточно книг, чтобы понять, как создавать симпатичные, хотя простые и статические, страницы для Всемирной паутины. Она овладела основами HTML, но не понимает и не хочет понимать в программировании ровным счетом ничего. Собственный сайт Бетси – мешанина модного дизайна, с размытыми шрифтами, лоскутами пастельных мазков, а также цитатами из Патти Смит и Эстер Дайсон.

Если Бетси требуется сложная обработка данных, ей приходится просить помощи у Эрни. Эрни – программист-спец нового поколения. Обожает компьютеры, компьютерные игры, компьютерные языки и компьютерное оборудование. В сравнении с программистами предыдущего поколения он пока немного легковесен: не знает языков C, C++, ассемблера, но невероятно умело обращается с инструментами вроде CGI, Perl, JavaScript и Visual Basic. Он знает сотни компонентов ActiveX и JavaBeans. Он способен создать достаточно функциональный модуль из сложных компонентов всего за несколько дней – та же работа в восьмидесятые годы заняла бы у программиста на C года четыре. Собственный сайт Эрни – случайная подборка цитат из «Звездного пути» и «Симпсонов». Здесь кричащий красный текст на черном фоне, восемь различных шрифтов, мигающий текст, потоковый звук, дергающиеся пиктограммы, кнопки Submit и ссылки на самые классные Quake-сайты.

Быстро стало понятно, что команда Elemental, не имея четкого представления о Бетси и Эрни, пыталась разработать программу, которая осчастливила бы обоих. Результатом стал коктейль мощных и сложных возможностей в графической оболочке. «Вы только посмотрите, какую замечательную штуку теперь может делать пользователь! – говорили они. Их „пользователь“ был размыт, и поэтому они понятия не имели о том, каковы его цели. Программисты в Elemental в целом симпатизировали Бетси, но по характеру им был ближе Эрни, а потому продукт естественным образом тяготел к потребностям именно Эрни.

Когда мы представили Бетси и Эрни, вся компания немедленно признала в обоих персонажах очень знакомые архетипы и ухватилась за них, как за полезные определения пользователей.

Кто кому служит

Визуальные среды для конструирования веб-сайтов были (и все еще остаются) горячим товаром на рынке, поэтому конкурентов было предостаточно, однако впервые наш клиент получил возможность оценить себя и своих конкурентов в сравнении с Бетси и Эрни.

Конкурирующие продукты также встали по разные стороны линии Бетси – Эрни. С одной стороны, несколько компаний создавали классные, новые инструменты для Эрни. Сложные и непростые в применении, эти инструменты позволяли Эрни создавать мощные, динамические, изощренные сайты для корпоративных клиентов.

С другой стороны, еще несколько компаний были заняты созданием классных новых инструментов для Бетси. Инструменты все как один были простые, наглядные, легкие в изучении, но слабые, словно котята. Они позволяли создавать только статические сайты с низкой функциональностью и без возможности работы с внешними базами данных.

Взглянув на поле битвы глазами Бетси – Эрни, мы все осознали, что здесь можно сделать сильный ход – дать Бетси инструмент, наделяющий ее более широкими возможностями. В результате Elemental создаст желанный продукт для еще не заполненного сегмента рынка. Программисты вскоре приняли имя «Бетси» в качестве боевого клича и сосредоточили усилия на том, чтобы ей помочь.

Для начала было неплохо, однако в ходе дальнейшего проектирования, в результате более пристального изучения целей Бетси, мы обнаружили интересную вещь.

В прежнем Мире, в мире простых, статических сайтов первого поколения, Бетси ни от кого не зависела. Она могла создать дизайн и собственно сайт для клиента, не прибегая к помощи Эрни. Бетси работала в знакомой области и могла сообщить потенциальному клиенту, каковы объемы работ, во что обойдется создание и когда работа будет сделана. Она могла быть уверенна, что выполнит свои обещания. Именно независимость и самостоятельность привлекли ее изначально ко Всемирной паутине, и поэтому она ушла с основной работы и стала работать на себя.

Возможности, предоставляемые Интернетом по мере его развития, росли очень быстро, и с такой же скоростью увеличивалась сложность создания сайтов. Сайты становились все мощнее, обретали дополнительную функциональность; появился прямой доступ к базам данных. Бетси уже не могла работать на столь низком уровне программирования. Кроме того, для нее программирование было совсем не столь привлекательным, и учиться она не хотела. Тогда она и встретила Эрни, способного решать за нее все технические проблемы. Он просто обожал все эти специальные штуки.

Но Бетси также обнаружила, что выполнение ею заказа теперь зависит и от Эрни. В каждом новом веб-проекте обязательно наступал момент, когда приходилось обратиться к Эрни, чтобы он обеспечил доступ к базе данных и создал код динамических страниц. Она уже не могла создать полноценный сайт без помощи Эрни, а Эрни – человек не столь пунктуальный, как Бетси. Она уже не могла сообщить клиентам дату готовности заказа и точно знать, что успеет в срок. Хаотичность действий Эрни мешала ее бизнесу. Так появилась немного измененная картина целей Бетси.

Бетси по-прежнему желала создавать классные, мощные, динамические сайты, но самая важная цель для нее стала иной. То, что было целью гигиенической и воспринималось Бетси как данность, но вскоре исчезло, получило теперь фокус внимания. Главной целью Бетси снова стала независимость, освобождение от Эрни. Она хотела иметь возможность начать отношения с клиентом, спроектировать, создать мощный, красивый, динамический сайт, функционирующий на основе базы данных, *и при этом не ждать, когда Эрни, наконец, разгадает какую-нибудь техническую головоломку.*

Изначально мы намеревались сделать инструменты Бетси более мощными, при этом сохранив простоту применения. Оставаясь весьма привлекательным, такой продукт всего лишь давал Бетси отсрочку в общении с Эрни, не позволяя ей достичь самой важной цели. Чтобы добиться успеха с Бетси, мы должны были спроектировать Drumbeat, позволяющий вовремя и самостоятельно завершать все проекты.

Эрни тоже не был абсолютно счастлив, работая с Бетси. Все, что он делал, требовало

одобрения Бетси, и она постоянно придиралась по поводу пиксела там, пиксела здесь – к тому, что, по его мнению, никакой роли не играло. Она требовала переделывать уже сделанное по пять-шесть раз, вносить ненужные, как он считал, изменения, прежде чем согласиться, что работа готова. Он хотел получить независимость от Бетси не меньше, чем она желала получить независимость от него.

Проектирование

На этом этапе мы смогли обосновать свою точку зрения. Вместо того, чтобы давать Бетси отсрочку в общении с Эрни, мы должны были возвести между ними непроницаемую стену и дать независимость обоим. Бетси по-прежнему будут нужны функции, которыми ранее ведал Эрни, и, в конце концов, Бетси для Эрни была неплохим источником дохода, поэтому их коммерческие отношения необходимо сохранить, тогда как их деятельность необходимо четко разграничить.

Строить стену между ними было необходимо на основе стандарта – интерфейса, позволяющего создавать и использовать функциональные модули. Эрни следует дать интерфейс программиста, позволяющий работать кодом, а Бетси – интерфейс дизайнера сайтов. Общей и нейтральной территорией стала бы программа Drumbeat. Эрни сможет писать мощные, гибкие модули и публиковать их при помощи функционального интерфейса Drumbeat. Бетси будет применять эти модули посредством интерфейса визуального программирования Drumbeat.

Теперь Бетси сможет создавать динамические сайты, работающие с базами данных, посредством опубликованных модулей, никогда не встречая автора этих модулей. Эрни сможет создавать, публиковать и продавать функциональный код, никогда больше не трогая цвет фона. Освободив эти два персонажа, мы помогли Бетси в области дизайна и производства, а Эрни – в области программирования.

Теперь Эрни выступает в роли автора инструментов, а не программиста, выполняющего заказы. Он создает подключаемые модули, которые могут становиться частью инструментария Бетси. У его модулей более широкая аудитория, поскольку он может продавать их многим другим Бетси, которые, в свою очередь, будут применять эти модули для многочисленных сайтов.

Случай интересен тем, что проектирование взаимодействия оказало значительное влияние как на внутреннюю структуру продукта, так и на позиционирование продукта на рынке. Это хороший пример, показывающий, как проектирование влияет на начинку, описывая всего лишь внешние проявления.

Откат

Разработчики Elemental поначалу решили, что наше решение не сработает, поскольку сразу представили себе ряд вырожденных случаев, когда Бетси все равно понадобятся

особые таланты Эрни. «Нельзя совсем исключать Эрни из цикла, – заявили они, – поскольку Бетси может понадобиться решить какую-то особую или сложную задачу».

Что ж, подумали мы, это справедливо, однако лишь в редких случаях. В большинстве же случаев она работает независимо, тогда как при текущем положении дел о независимости и мечтать не приходится. В этих редких вырожденных случаях она просто будет возвращаться в прежнюю ситуацию зависимости от Эрни. Это определенно не ухудшает состояние дел, а в большинстве случаев заметно улучшает.

Независимость важна для Бетси, и ради ее достижения она готова на соразмерные жертвы. Drumbeat позволяет ей создавать полноценные сайты без помощи Эрни, и это примиряет ее с небольшими компромиссами в дизайне, открывающими доступ к готовым программам Эрни[33]. Жертва невелика, поскольку запросы большинства клиентов укладываются в стандартные схемы. Если она получит контракт на создание корпоративной сети для универмагов Wal-Mart или системы оперативного резервирования для отелей Hilton, то определенно задействует человека, обладающего талантами разработчика, чтобы он помог справиться с этими монументальными задачами, но в большинстве случаев это не требуется.

Прочие моменты

Изначально в программе было множество плавающих панелей с разнообразными инструментами для рисования, и каждая заслоняла часть макета создаваемого сайта. В Elemental все свыклись с идеей, что пользователям нравится гонять панельки по экрану в процессе работы. При каждой демонстрации продукта разработчики гордо хвастались этими панелями.

Участники нашей команды проектировщиков в один голос заявили, что панели назойливы, сложны и совершенно излишни. Разумеется, необходимо обеспечивать доступ к инструментам, но мы знали и более привлекательные способы. Однако каждый раз, когда мы отрицательно отзывались о панелях, программисты (и руководители разработки) тут же заявляли, что панели всем очень нужны.

Наблюдая, как реальные Бетси работают с продуктом, мы скоро поняли, почему плавающие панели были настолько популярны. Изначально программа была спроектирована так, что без панелей было не обойтись. Большинство инструментов каждой панели редко находили применение, но при этом на каждой панели было по меньшей мере два очень полезных и востребованных инструмента. То есть Бетси нужны были все панели для решения самых простых задач. Из-за лишних инструментов каждая панель была чересчур велика, и все панели плавали поверх изображения сайта, находящегося в разработке, поэтому, работая над сайтом, Бетси постоянно приходилось перемещать панели. Альтернативный режим позволял зафиксировать панели вдоль одной из границ экрана, но это означало лишь, что Бетси придется постоянно прокручивать изображение сайта. Бетси попала в безвыходное взаимодействие. Она могла терять время на ненужную прокрутку или на перемещение панелей. Такие ненужные для пользователя действия мы называем «акцизами», и изначально программа была ими насыщена.

Мы понимали, что под рукой должны быть только инструменты, применяемые часто, и для решения проблемы все инструменты лучше поместить в одно место. Иначе Бетси запутается.

Путем простой реорганизации и отсеивания нечасто востребуемых функций мы значительно уменьшили размеры панелей. После этого мы закрепили панели в определенных точках. Так панели стали практически незаметной составляющей интерфейса. Вот хороший пример, показывающий, как целеориентированное проектирование снижает объем кода, необходимого для создания интерфейса.

[33] Создание сайтов – это программирование, и Бетси точно так же не способна сопротивляться очарованию повторного использования кода.

* * *

Как проектирование, так и продукт стали успешными. Ближе к завершению работ над версией, основанной на наших исследованиях, Elemental удалось получить значительное венчурное финансирование, не в последнюю очередь благодаря инновациям во взаимодействии. После выхода в свет Drumbeat получил многочисленные положительные отзывы в отраслевой прессе. Эта цитата из журнала «PC» вполне отражает ситуацию.

> «Drumbeat – продукт уникальный и впечатляющий, его возможности автоматизации сложных веб-задач превосходят все другие присутствующие на рынке решения. Продукт позволяет непрограммистам решать задачи очень просто. Вы сможете создавать огромные профессиональные сайты и даже применять технологию Active Server Pages, не написав и строчки кода.»

Этот продукт стал успехом несмотря на то, что многие другие программы для создания сайтов пробились на рынок раньше.

* * *

Как видите, взгляд на вещи через призму пользовательских целей может дать уникальную и обладающую невероятным потенциалом перспективу, открывающую новые возможности для творчества. Это и есть основа целеориентированного проектирования.

Глава 11
Проектирование для людей

В предшествующих главах я описал персонажи и подчеркнул превосходство целей над задачами. Только если у нас есть персонажи и мы представляем себе их цели, мы можем начать изучение задач без боязни, что они помешают процессу проектирования. Свой инструмент для рассмотрения задач мы называем «сценариями». Сценарий – это сжатое описание способов применения программного продукта персонажем для достижения цели. Эта глава посвящена сценариям, а также ряду других полезных инструментов проектирования. Она содержит и примеры работы этих инструментов, в частности сценариев в реальных проектах.

Сценарии

По мере детализации проектирования эффективность сценариев повышается. Наши персонажи проходят через эти сценарии, словно актеры, читающие роли, что позволяет проверять корректность проектирования и выдвинутые предположения. Неудивительно, что наш сценарный процесс часто сравнивают с игрой по Станиславскому – актер должен вжиться в роль, знать то, что знает персонаж, чувствовать то, что чувствует персонаж. Мы стараемся думать так, как думает он. Мы забываем о собственном образовании, способностях, подготовке, инструментах и думаем, что обладаем лишь данными и биографией персонажа. Поскольку мы проектировщики, а не актеры, без контекста и конкретных деталей может быть нелегко, поэтому сценарии оказываются большим подспорьем. Зная, к примеру, что Бетси пытается создать сайт для страховой компании, нам проще ставить себя на ее место. И это не так странно, как может показаться. Ведь программисты ставят себя на место своих компьютеров. Программисты привычно описывают действия компьютера от первого лица, они говорят: «Я обращаюсь к базе

данных, а затем сохраняю записи в буфере». Человек говорит «я», но всю работу на самом деле выполняет компьютер. В роли компьютера человеку проще симпатизировать потребностям машины в процессе написания кода.

Сценарии создаются на основе информации, собранной в ходе первой фазы – исследования. Обычно в ходе интервью и непосредственного наблюдения за пользователями удается много узнать об их задачах. Цели стабильны и неизменны, задачи же неустойчивы, подвержены изменениям и часто оказываются ненужными в компьютеризованных системах. В процессе разработки сценариев мы стараемся находить и вычеркивать задачи, существование которых обусловлено лишь исторической необходимостью.

Эффективность сценария определяется в большей степени его охватом, чем глубиной. Иначе говоря, важнее, чтобы сценарий описывал процесс от начала до конца, чем чтобы он описывал каждый шаг в исчерпывающих подробностях.

Важно развивать лишь те сценарии, которые позволяют продвигаться вперед в процессе проектирования и не плутать в дебрях исключительных ситуаций. Мы разрабатываем лишь два вида сценариев, хотя сценариев каждого вида может быть и несколько. Это сценарии повседневные и сценарии обязательные.

Повседневные сценарии

Повседневные сценарии – самые полезные и важные. Они описывают основные действия, которые пользователь выполняет чаще всего. Так, для системы сопровождения программных ошибок типичные повседневные сценарии – поиск записей о дефектах и заполнение форм о свежевыявленных дефектах. Любой сотрудник технической поддержки выполняет эти две задачи ежедневно и по многу раз.

В общем случае для большинства пользователей репертуар повседневных сценариев оказывается весьма ограниченным. Чаще это один или два сценария. И редко их бывает больше трех.

Повседневные сценарии нуждаются в самой надежной поддержке качественным взаимодействием. Новые пользователи должны быстро овладевать такими сценариями, а значит, требуется качественная поддержка быстрого обучения. То есть инструкции по применению должны быть написаны у программы «на лбу». Однако частота повторений вскоре приводит к ненужности инструкций, поэтому у пользователей быстро появляется потребность в ускоренных методах работы, таких как клавиатурные сокращения. Кроме того, по мере приобретения опыта пользователи начинают ощущать потребность в подгонке повседневных взаимодействий под индивидуальный стиль работы и личные предпочтения.

Обязательные сценарии

Обязательные сценарии описывают все действия, выполняемые и нечасто, но неукоснительно. Очистка баз данных и создание исключительных запросов могут оказаться именно в этой категории сценариев.

Обязательное взаимодействие также требует поддержки механизма обучения. Однако пользователь никогда не перейдет на более высокий уровень прохождения таких сценариев. Редкое применение означает, что любой пользователь согласится с механизмами, предложенными программой, поэтому индивидуализация не потребуется. Это освобождает разработчиков от необходимости обеспечивать тот же уровень доводки, которого требует повседневный сценарий. Примерно так же роскошный интерьер салона нового Ягуара и отличается от грубого металла его подкапотного пространства.

В большинстве продуктов репертуар обязательных сценариев достаточно ограниченный, но количество последних обычно превышает количество сценариев повседневных.

Сценарии исключительных ситуаций

Разумеется, существует третий вид сценариев – сценарии для исключительных ситуаций. Программисты, естественно, обращают особое внимание именно на них, но в процессе проектирования продукта эти сценарии можно игнорировать. Это не означает, что соответствующей функциональности нет места в программе, но означает, что взаимодействие для таких сценариев можно проектировать грубо и упрятывать в глубины интерфейса. Работоспособность *кода* может зависеть от того, обрабатываются ли исключительные ситуации, а успех *продукта* зависит от способности справляться со случаями, описанными в сценариях повседневных и обязательных.

Если пользователь решает некую задачу часто, то взаимодействие в данном месте программы должно быть продумано до мелочей. Точно так же, если задача выполняется редко, но в любом случае, то взаимодействия для этой задачи, пусть и преследующее иные цели, должно быть спроектировано качественно. Задачи, не являющиеся обязательными или повседневными, просто не требуют тщательного проектирования. Времени и денег всегда не хватает, поэтому такие задачи – хорошая возможность сэкономить ресурсы и потратить их на то, что приносит больше пользы. Мы должны создать все сценарии, но тщательно проектировать интерфейс следует лишь для сценариев важных или применяемых постоянно.

* * *

Персонажи, цели и сценарии – тяжелая артиллерия проектировщика. Прежде чем перейти к примеру применения сценариев, поговорим о некоторых других полезных понятиях проектирования: адаптирующемся интерфейсе, вечных середняках, словаре, мозговом штурме и нестандартном мышлении.

Адаптирующийся интерфейс

Взаимодействие всегда можно упростить – достаточно удалить функции и сделать продукт менее функциональным. В большинстве случаев такая тактика неприемлема. Более сложные задачи проектирования требуют простоты применения продукта без принесения в жертву функциональности и мощи. Это сложно, но выполнимо. Здесь требуется лишь техника, которую я называю адаптирующимся интерфейсом.

Программа должна предоставлять массу функций, но в конкретный момент времени конкретному пользователю для конкретной задачи нужны далеко не все функции. Для каждого сценария персонажу требуется лишь небольшое подмножество органов управления и данных, хотя этот набор может меняться с течением времени и при изменении решаемой задачи. Интерфейс станет на порядок проще, если органы управления и данные, необходимые для повседневных сценариев, будут бросаться в глаза, тогда как все прочие элементы интерфейса отойдут на второй план, за пределы поля зрения.

Интерфейсы большинства крупных программ очень похожи на меню в китайских ресторанах, где каждая страница испещрена сотнями пунктов. Возможно, для ужина именно это и нужно, но в продуктах высоких технологий только мешает.

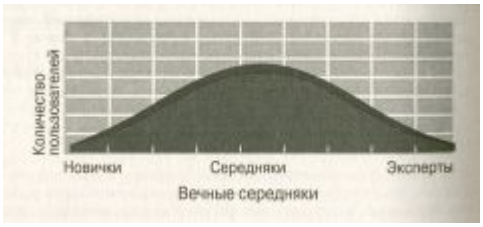
В Microsoft Word, к примеру, стандартная панель инструментов содержит пиктограммы для загрузки документа, закрытия и печати текущего документа. Перечисленные задачи выполняются с разумной частотой, и присутствие пиктограмм оправданно. Однако здесь же, рядом, располагаются пиктограммы, генерирующие схему документа и внедряющие в документ электронные таблицы. Microsoft поместила эти пиктограммы в Основном рабочем пространстве, чтобы мы смогли оценить мощь Word. К несчастью, большинство пользователей к этим функциям так никогда и не обращаются, а если и обращаются, то вряд ли часто. Этим функциям просто не место на панели инструментов, потому что панель инструментов – это идиома интерфейса, обеспечивающая доступ к самым востребованным

функциям.

Вечные середняки

Обычно самые мощные инструменты позволяют нам понять, представить личности наших пользователей, вжиться в них. Одну умозрительную модель мы применяем постоянно – она называется «вечные середняки». Большинство пользователей – не новички и не эксперты, они просто вечные середняки. Помните Рупака, Шэннон, Декстера и Роберто из главы 9, где речь шла об уровне подготовленности? Квалификации этих людей совершенно различны, но все четверо являются вечными середняками.

Опыт людей, работающих с интерактивными системами, как и многие другие характеристики, тяготеет к классическому «колоколу» нормального статистического распределения. Если построить график зависимости количества пользователей любого электронного продукта от уровня их профессионализма, то получится немного новичков в начале шкалы, немного профессионалов ближе к ее концу и преобладание пользователей среднего уровня по центру.



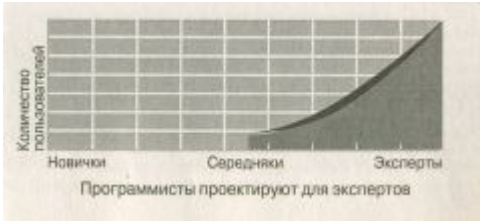
Но статистика не дает полной картины. Это моментальная фотография, в которой нет динамики, и, хотя большинство – вечные середняки – остаются обычно в своей категории длительное время, люди на концах кривой – новички и эксперты – постоянно меняются. Сложность сохранения высокого уровня профессионализма означает, что специалисты быстро появляются и быстро исчезают. Новички, в левой части кривой, сменяются еще быстрее.

Какое-то минимальное время *любой* пользователь продукта проводит в статусе новичка, но *никто* надолго в этом статусе не задерживается. Просто потому, что *никто не любит* быть новичком, ни у кого нет такой цели. Людям не нравится собственная некомпетентность, а новички некомпетентны по определению. И напротив, обучение и совершенствование – естественный, увлекательный, приносящий удовольствие процесс, поэтому новички очень быстро превращаются в середняков. К примеру, учиться играть в теннис увлекательно, но первые несколько часов или дней неудачные попытки отбить мяч приводят в отчаяние. Овладев базовым навыком, вы уже не тратите все время на беготню за пропущенными мячами, а действительно продвигаетесь вперед. Состояние новобранца попросту никого не привлекает, и каждый человек быстро выходит из него, попадая в некое подобие средней лиги. Если через несколько дней вы по-прежнему бессистемно гоняетесь за теннисными мячами, то забросите теннис и займетесь рыбной ловлей или станете филателистом.

Новички на левом конце кривой либо мигрируют в центральную область, где обитают середняки, либо попросту исчезают с графика и находят другой вид деятельности, где *способны* превратиться в середняков. Однако популяция центральной части графика очень стабильна. Достигая адекватного уровня, люди, как правило, остаются в этой области навсегда. Особенно это верно для продуктов, обладающих высоким когнитивным сопротивлением: пользователи не получают удовольствия от их изучения. Поэтому они изучают необходимый минимум, после чего учеба прекращается. Лишь *хомо логикус* получает удовольствие от изучения сложных систем.

Теперь сопоставим «колокол» кривой статистического распределения с тем, как

разрабатывается программное обеспечение. Все программисты относятся к разряду экспертов, поскольку им приходится исследовать все, даже самые странные варианты и маловероятные ситуации, чтобы создать код для их обработки. Естественная склонность к проектированию на основе собственных предпочтений означает, что они создают код по модели реализации, назначая всем функциям равный вес во взаимодействии. Если построить график пригодности типичного продукта, созданного по модели реализации, то наивысшая его точка будет расположена в правой части шкалы, где обитают эксперты. О пользователях среднего уровня никто особенно не заботится.



В пределах компании руководство, отделы продаж и маркетинга постоянно расхваливают продукт клиентам, журналистам, партнерам, инвесторам, то есть людям, с продуктом не знакомым. Эти профессионалы вынуждены постоянно контактировать с начинающими, поэтому их видение пользовательской аудитории искажено и определяется именно этой проблемной группой. Все эти влиятельные участники процесса естественным образом лоббируют упрощение интерфейса на благо новичков. Они желают внедрить в продукт дополнительные средства обучения, облегчающие начинающим жизнь. Теперь наивысшая точка графика пригодности продукта к взаимодействию оказывается в левой части шкалы, где обитают новички.

Наложив два графика, мы увидим, что сильнейшее влияние на проектирование взаимодействия оказывают антиподы, а самое главное, что ни те, ни другие не достигают цели. Программисты требуют взаимодействия, уместного только для специалистов, тогда как маркетологи требуют взаимодействия, уместного только для новичков, тогда как самую большую, самую стабильную и самую важную группу пользователей – вечных середняков – просто игнорируют.

Такой разрыв между образом пользователей, сложившимся у разработчиков, и действительной природой пользователей создает дополнительное когнитивное сопротивление. Вы легко увидите это в большинстве внутренних корпоративных приложений и в большинстве массовых продуктов, основанных на программном обеспечении. Чтобы успешно применять эти продукты, необходимо быть программистом, но в то же время эти продукты изобилуют артефактами вроде мастеров и оперативной справки

для начинающих. Эти возможности и есть дополнительно приделанные средства обучения. Мастеры и справка, как правило, выручают пользователей в определенных ситуациях, не обучая тому, как в будущем не попадать в неприятные ситуации. Специалисты мастерами и справкой никогда не пользуются, а новички стремятся побыстрее избавиться от этих оскорбительных напоминаний о невежестве. Однако вечные середняки навсегда остаются в компании этих артефактов.

* * *

Вооруженные инструментами целеориентированного проектирования, такими как персонажи, цели, сценарии, вечные середняки, адаптирующийся интерфейс и другими, мы можем уверенно штурмовать проблемы проектирования, представленные нашими клиентами. Мы знаем, что даже самые неподатливые задачи в конечном итоге сдаются под натиском нашего процесса.

Представь себе!

Каждый инженер представляет свой продукт в собственных терминах, но из-за необходимости программировать редко видит продукт в контексте конкретного пользователя (по крайней мере, конкретизация не достигает уровня, который я нахожу полезным). Мозговые штурмы позволяют нам избавиться от всех ограничений и ожиданий и начать проектирование с чистого листа, уделяя большое внимание персонажам и их целям. Мы часто проделываем упражнение на творческое мышление. Это упражнение называется «представь себе!» и заключается в прохождении сценария с «волшебным компьютером», не имеющим никаких ограничений.

Это упражнение подчеркивает контраст между задачами и целями. Задачи обычно меняются вместе с технологией, тогда как цели остаются неизменными. Воображая волшебную технологию, мы принудительно изменяем все задачи, обнажая истинные цели. Несмотря на кажущееся притворство, процесс этот является весьма конкретным умственным упражнением. Иногда проектировщиков осеняет верным ответом, но не реже им приходится долго дискутировать и изучать проблему, прежде чем получить этот ответ.

Словарь

В процессе проектирования и особенно в ходе мозговых штурмов я отдельно подчеркиваю необходимость создания и применения подробного и точного словаря. Я считаю, что технический нюанс проектирования интерактивных продуктов настолько важен, что единственное неправильно истолкованное слово может стать причиной краха целого проекта. Мне приходилось наблюдать, как разные участники команды клиента применяют распространенные слова вроде «кнопка» или «диалог» для обозначения качественно различных вещей. Мне вспоминается встреча с заказчиком, на которой десять высокооплачиваемых специалистов в течение двух часов ожесточенно спорили из-за разногласия, возникшего лишь потому, что различные участники пользовались различными определениями одних и тех же понятий.

Если нет слов для выражения идеи, то ее практически невозможно передать. И определенно невозможно такую идею проанализировать и разложить на составляющие до достаточного уровня технических деталей, чтобы реализовать ее на языке C# или Java.

Если слова не точны, программисты рефлективно ищут спасения в самом точном из доступных методов выражения: исходном коде. И хотя не существует ничего более точного, чем код, также не существует и ничего более крепкого и неподвластного изменениям. Поэтому часто путаница в терминологии заставляет программистов преждевременно начинать создание кода, и этот код становится проектированием де-факто, независимо от

того, насколько такое проектирование уместно или корректно.

Если терминов недостаточно или они определены нечетко, люди начинают мыслить консервативно. Без хорошего набора точных терминов новые идеи невозможно защищать на должном уровне, и в результате идеи отметаются раньше времени.

Наши термины не имеют ничего общего с теми, которые будут напечатаны на упаковке продукта. Наш словарь предназначен для внутреннего употребления, поэтому нас не заботит красота слов с точки зрения маркетолога. Слова должны быть всего лишь точны. Позже отдел маркетинга придумает слова, подходящие для завлечения покупателей. Продукт ScanBank компании Logitech изначально назывался «верстаком», это слово идеально подходило для нашего процесса проектирования и никогда не предназначалось для широкой публики.

В одном проекте наши проектировщики зашли в тупик. После долгих споров стало очевидно, что существуют расхождения в трактовке терминов. Наша дискуссия была неэффективна, потому что не было общего словаря. Я настоял на том, чтобы разбить проект на отдельные фрагменты, с которыми мы все согласимся, и дать фрагментам совершенно новые названия, не относящиеся к делу. Без особой причины я выбрал названия горных массивов Аляски. Четыре основных фрагмента продукта мы назвали «гора Святого Ильи», «Брукс», «Аляска» и «Рангелл». Мы хорошо посмеялись над неуместностью новых терминов, но после этого практически сразу достигли согласия и быстро стали продвигаться вперед.

Языковой прорыв

Прежде всего, хороший словарь делает общение более эффективным. Однако создание терминологии иногда имеет и другой очень важный эффект. Время от времени нам приходится работать с командами, в которых определенные термины уже стали частью культуры. Хороший пример – фраза Microsoft «Embrace the Internet». Подобные фразы и слова могут иметь почти религиозное значение и произноситься с оттенками благоговейного трепета. Такое благоговение приводит к неспособности разобрать значение слова и повторно изучить его в свете новых императивов проектирования. Означает ли фраза, что следует идти навстречу браузерам, или же HTML, или же только протоколам TCP/IP? Эти священные слова – забор вокруг храма. Растоптав священные верования клиента, мы не продвинемся в процессе проектирования. Поэтому мы разбиваем процессы, задачи, программы на четко определенные обособленные фрагменты и назначаем им новые имена, не имеющие унаследованного смысла. Эти новые имена, обычно еще и шутливые, и такое легкомыслие позволят «пробить» серьезные мины участников.

Реальность смеется последней

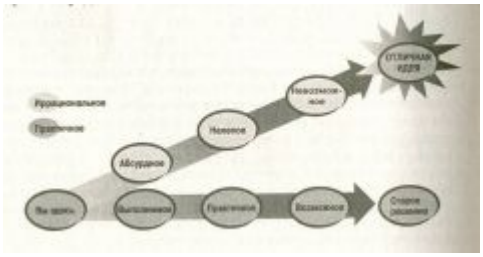
Типичный процесс разработки продукта начинается с перечисления ограничений и условий. Катехизис того, что «мы не можем сделать», повторяется достаточно часто и убедительно, чтобы стать доктриной независимо от того, насколько этот катехизис соответствует истине. Проектировщики взаимодействия должны со здоровым скептицизмом относиться к предположениям о невозможности чего-либо. Раз за разом мы находим способы обходить предполагаемые ограничения только потому, что отказываемся воспринимать их всерьез.

Разумеется, встречаются и настоящие ограничения, которые обойти действительно невозможно, однако в попытках это сделать приобретается ценный опыт. Даже если мы не можем изящно обойти ограничение, наше путешествие по тупиковому маршруту может пролить свет на какую-то не замеченную ранее возможность. Этот процесс основан на работе Эдварда де Боно, посвященной нестандартному мышлению[34].

[34] Edward de Bono «Lateral Thinking, Creativity Step by Step» (Нестандартное мышление: Творчество шаг за

Программисты – короли практичности. Прагматизм практически лишает их терпимости к фантазиям. Но эта сила может стать и слабостью, поскольку случается, что практичный подход не позволяет решить задачу. Изобретая, инженеры находят решение путем последовательного изучения практичных, возможных шагов. Как следствие, их решения всегда оказываются производными старых решении, а этого часто недостаточно.

Мы же просто предполагаем, что возможно все, и проектируем, исходя из этого убеждения. Обходя предполагаемые ограничения, мы видим цели и персонажи более отчетливо и находим решения, до которых невозможно добраться традиционными путями.



Инженеры испытывают тревогу, отступая от твердых рациональных основ, а потому предпочитают мириться с предполагаемыми ограничениями. Они *знают,* что мы в конечном итоге встретимся с этими ограничениями, а потому считают себя обязанными защищать их. Они называют это «ролью адвоката дьявола». Все это очень хорошо, но в чем окружающая действительность не нуждается, так это в том, чтобы ей помогали создавать трудности. *Реальности не нужны адвокаты,* поскольку от нее возможно отречься. Реальный мир всегда смеется последним. Зная, что реальность всегда найдет подходящий ответ, мы понимаем, что невозможное никогда не станет реальным, и здесь не спасают ни воображение, ни проектирование. Лишь человек, не заинтересованный кровно в успехе предприятия, может спроектировать что-то, что «невозможно» создать. Более того, мы часто обнаруживаем, что ограничения иллюзорны и надуманы. И это нельзя понять, не обойдя их.

Пример: Logitech Scanman

Наш инструмент «представь себе!» оказался особенно полезным в одном крупном проекте. Подразделение корпорации Logitech, занятое производством сканеров, пригласило нас для участия в проектировании программного обеспечения для совершенно нового поколения настольных сканеров, ориентированных на рынки домашнего и малого офисов.

В новом сканере Logitech под кодовым названием «Peacock» были применены технологии сканирования нового поколения, а само устройство подключалось к компьютеру через новый порт USB. Этот недорогой продукт, по форме и размерам напоминающий свернутую газету, удобен и не занимает много места на столе. В прорезь сканера можно вставить любую страницу, после чего небольшой двигатель протягивает страницу через сканер, при этом происходит считывание изображения.

шагом), 1970, Harper & Row, Publishers, New York.

Компания Logitech уже давно сосредоточила усилия на выпуске небольших дополнительных аппаратных компонентов, особую ценность которым придает сопровождающее программное обеспечение. Звучит определенно неплохо, с точки зрения инженеров Logitech. Однако с точки зрения пользователя подход не очень приятный. Потому что не ориентирован на конечные цели.

Logitech предполагала, что многочисленные программные возможности увеличивают ценность аппаратного устройства. В конце концов, считалось, что добавление возможностей в программу гораздо дешевле добавления возможностей в аппаратную часть. Такая логика рассматривает уравнение стоимость-выгода с точки зрения производителя, а не пользователя.

Предшественник продукта Peacock ломился от возможностей, и у каждого участника команды, будь то маркетолог, руководитель, программист или менеджер продукта, были свои любимые возможности, за которые он активно боролся на совещаниях. Но если когда-либо существовал продукт, нуждающийся в вивисекции, то это был именно Peacock.

Мы редко считаем необходимым вырезать из продукта функции, чтобы сгладить неровности взаимодействия. Однако в случае Peacock широко распространенная идея о том, что многочисленные возможности увеличивают ценность продукта, явно была ошибочной. Наши персонажи и сценарии отчетливо показали, что интерфейс продукта перегружен ненужными, не востребованными и не находящими применение возможностями.

Как обычно, мы начали процесс с создания набора персонажей. Расскажу о том, как мы сконструировали этот набор.

Магазинная цена сканера была около $150. Для массового продукта это был достаточно мощный сканер с высоким разрешением и цветопередачей, однако до уровня профессиональных планшетных сканеров, стоивших обычно от $800 до $1000, он не дотягивал[35]. Всем было ясно, что основной рынок сбыта продукта формируют пользователи в домашних и малых офисах, известных под собирательным называнием SOHO (small office, home office).

Малкольм, боец фронта Всемирной паутины

Для представления сегмента пользователей SOHO мы создали персонаж Малкольма, бойца фронта Всемирной паутины. Этот молодой человек открыл на дому небольшую фирму по созданию сайтов. Он не разбирается глубоко в технических вопросах и не является дизайнером, однако знаком с компьютерами и знает, что оптимизированные изображения гораздо быстрее скачиваются из Интернета, чем здоровые, полноцветные. Сканер Peacock позволяет ему легко получить изображения среднего разрешения для сайтов, не неся излишних расходов и не вдаваясь в детали процесса.

Чед Марчетти, мальчик

Сканер Peacock был привлекателен и для владельцев домашних компьютеров, которые сканировали картинки и документы для личных, а не деловых целей. Для представления домашних пользователей мы изобрели персонаж Чеда Марчетти, десятилетнего мальчика, оформляющего при помощи сканера свои домашние задания.

DPI Магнум

Мы знали, что профессиональным дизайнерам нужны тысячедолларовые планшетные сканеры, а потому снизили свое внимание к этому сегменту рынка. Но мы знали также, что

[35] Как обычно, время и пикирующие цены на продукты из кремния значительно изменили поле битвы сканеров, но в 1997 году ситуация была именно такая.

этот сегмент нельзя просто игнорировать, ведь и из искры может разгореться пламя. У начинающего дизайнера, еще не имеющего постоянной работы, нет свободных денег, поэтому Peacock сгодится на первый год или два, пока он не сможет позволить себе сканер профессионального уровня, однако только в том случае, если Peacock поможет добиться достаточной производительности.

Искрой стал персонаж по имени DPI Магнум. (Вариация на тему старого телешоу Тома Селлека «Magnum, P.I.[36]» и аббревиатуры слов dots-per-inch, распространенной меры разрешения оцифрованного изображения.) Магнум представляет, возможно, не самый крупный сегмент пользователей, но человек он, несомненно, влиятельный. Все его друзья, владельцы домашних компьютеров, просят его совета, когда речь заходит о программном и аппаратном обеспечении для работы с графикой. Через год он сможет позволить себе планшетный сканер, но до тех пор обойдется аппаратом Peacock.

Малкольм и Чед не особенно разбираются в обработке изображений. Малкольм слишком сосредоточен на другом – ему нужно создавать сайты и зарабатывать деньги. У Чеда тоже иные интересы – не потерять изображения в глубинах файловой системы. Ни тот, ни другой не видят серьезных причин ковыряться в пикселах. Обоим требуется сканировать изображения, обрезать их, а затем внедрять в документы. Конечным результатом являются именно эти документы, а никак не изображения. Мы обнаружили, что три важных цели Малкольма и Чеда совпадают:

Они не желают разбираться в сканерах, разрешениях, настройках.

Они желают быстро и легко находить на своем компьютере уже отсканированные изображения.

Они желают легко и быстро внедрять отсканированные изображения в другие документы при помощи других программ.

DPI Магнум гуляет сам по себе: он хорошо знает, что такое разрешение и чувствует себя, как рыба в воде, манипулируя различными настройками. Таким образом, можно предположить, что включение подобной функциональности в продукт пойдет Магнуму на пользу. Однако Магнум уже приобрел Adobe Photoshop. Эта мощная, сложная, дорогая программа обработки изображении – его основной инструмент, и он отлично им владеет[37]. Для решения всех своих задач (не имеет значения, насколько мелких) он применяет Photoshop. Любая попытка продублировать функциональность и мощь Photoshop в рамках Peacock обречена. Как и Пи Герман[38], вышедший на ринг против Джорджа Формана, Peacock не продержался бы и раунда против чемпиона. Нет смысла вкладывать усилия в то, что не найдет применения и станет позорным клеймом.

Однако же в двух случаях из трех цели Магнума идентичны целям Чеда и Малкольма: он желает легко находить свои изображения и легко переносить эти изображения в другую программу (Photoshop).

Лишь во время сканирования Магнуму может понадобиться указать физическое разрешение в точках на дюйм. В старых, более медленных сканерах понижение разрешения позволяло экономить время при сканировании, что Магнум и делал. Новый Peacock гораздо быстрее и дает максимальное разрешение, равное вполне приличным 200 dpi. При полноцветном сканировании процесс занимает около 20 секунд для формата A4. Потратив десять секунд на изменение режима, Магнум сэкономит лишь пять секунд при сканировании, но получит при этом более низкое качество, так что овчинка выделки не стоит. При

[36] Частный детектив Магнум. – *Примеч. перев.*

[37] Как бы мне хотелось перепроектировать взаимодействие в этой мощной сложной программе! Примечание: по меньшей мере шесть человек, читавших рукопись, подчеркнули эту сноску и добавили комментарии вроде: «И мне тоже!» и «Пожалуйста, сделай это!»

[38] Pee Wee Herman – псевдоним американского комика Пола Ройбенса. – *Примеч. ред.*

достаточно высокой скорости сканирования придет ли в голову кому-нибудь – даже Магнуму – уменьшать разрешение? Такое проникновение в суть вещей позволило нам понять, что цели всех троих не противоречат друг другу, так что мы можем осчастливить пользователей и при этом избавиться от большинства ненужных возможностей продукта.

Игра «Представь себе!»

В ходе мозговых штурмов мы сыграли в игру «Представь себе!». Мы обнаружили, что Чед вполне доволен возможностью помещать изображения в компьютер, вообще не пользуясь сканером. Это упражнение показало, что ни Чеду, ни Малкольму, ни Магнуму не хочется разбираться с аппаратной частью процесса. С этого ракурса было легко увидеть, что Чеда интересовало только отсканированное изображение, причем после того, как оно уже попало в компьютер. Его не волнует, если изображение сканируется при помощи черной магии, однако интересует возможность потом найти это изображение, обрезать его и открыть в других программах.

Большинство продуктов конкурентов, в том числе и продукт, предшествовавший Peacock, просто сбрасывали изображения в файловую систему, оставляя пользователя наедине с традиционной иерархией, позволяющей хранить и находить отсканированные изображения. Но этой файловой системой в действительности очень сложно пользоваться, она практически бесполезна.

Файловая система требует, чтобы Чед назначил имя отсканированному изображению, затем выбрал место в иерархии файловой системы, где изображение следует сохранить. А если Чеду понадобится найти изображение, ему придется вспомнить имя изображения и место, где это изображение хранится. Так уж получилось, что Чед не очень силен в запоминании подобных фактов. А компьютер, обладая жестким диском, идеально подходит для запоминания подобных фактов, но не делает этого. Он заставляет Чеда помнить и место хранения, и имя файла.

В нашем варианте программное обеспечение сканера не заставляет Чеда назначать изображениям имена и место сохранения. Программа все делает сама. Когда Чеду понадобится найти изображение, он сможет воспользоваться любым из его свойств, таких как дата сканирования, размер, содержание, отметка об экспорте в другую программу.

Не позволяя Чеду и Магнуму возиться с различными аппаратными настройками сканера, мы сосредоточили усилия на трех более важных моментах:

• Исключили из интерфейса все идиомы управления сканером.

• Сделали невозможной потерю отсканированных изображений в дебрях файловой системы.

• Предельно облегчили перенос отсканированных изображений в документы других программ.

Мы рассмотрели все доступные функции работы с изображениями и решили, что лишь три из них абсолютно необходимы. Остальные можно исключить или позднее выполнить в других, более подходящих приложениях, таких как Photoshop. Вот эти три функции:

• Обрезка, отсечение изображения по краям.

• Изменение размера изображения.

• Изменение ориентации, поворот изображения.

Набор функций получился очень маленький, но в него вошли только необходимые и часто используемые возможности, поэтому мы решили обеспечить всем функциям высочайшее качество и простоту применения. На коде удалось сэкономить в целом столько, что команда разработчиков получила необходимое время на достижение поставленных целей.

Высококлассная обрезка

Все компьютерные инструменты обрезки, с которыми мне приходилось сталкиваться, работают одинаково неправильным образом. Пользователь щелкает мышью и растягивает прямоугольный контур. Щелчок происходит в левом верхнем углу области отсечения, а точка, где заканчивается движение, становится правым нижним углом области. Все, что находится вне области, удаляется, а фрагмент внутри области становится новым изображением. Метод быстрый, простой, легкий в реализации, доступный для понимания. Тяжеловесная графическая программа Photoshop тоже использует этот способ. Тем не менее, способ имеет серьезные недостатки. Прежде всего, он дает низкую точность – область следует выделять одним четким движением. При этом весьма вероятно, что, определив три стороны области, пользователь не сможет определить верно четвертую, не затрагивая одну или несколько корректно размеченных сторон. Кроме того, необратимость операции означает, что программа может сохранить два различных варианта обрезки одного изображения.

Наш вариант инструмента решил обе проблемы простыми для освоения и понимания

способами. На каждой из четырех сторон отсканированного изображения присутствует постоянно видимый якорь линии обрезки. Якорь является наглядным инструментом непосредственного манипулирования. Теперь Чеду достаточно щелкнуть по якорю и потащить за него, чтобы получить мгновенный и соответствующий действию наглядный отклик, оценить последствия своих действий. По мере перемещения якоря часть изображения, оставленная «за бортом», меняет свой цвет на призрачно-серый. Становится очевидно, что происходит обрезка изображения, но также ясно, что обрезка еще не произошла необратимо. Чед может так же легко перетащить якорь на прежнее место и таким образом восстановить фрагмент изображения.

Перемещая один якорь, Чед сразу понимает, что четыре стороны области обрезки независимы, что перемещение одной стороны не затрагивает три другие. Он может корректировать и изменять область обрезки, пока не получит нужный результат. Совсем другое ощущение оставляют традиционные инструменты обрезки, где процесс модален, необратим, должен выполняться одним идеальным движением. Очень немногие пользователи компьютеров обладают нужной ловкостью рук, позволяющей выполнить хорошо такое движение. И Чед определенно не входит в их число. Более того, обрезка должна быть наглядной и, как правило, итеративной. Даже художникам требуется несколько попыток, чтобы добиться нужного результата. Старые инструменты просто не поддерживали такой подход. А тот, что мы создали для Logitech, делал это замечательно.

Даже окончательный выбор Чеда не делал обрезку необратимой. Текущие настройки области обрезки считались просто свойством изображения, а изображение всегда хранилось в полном объеме (отдельный пункт меню позволял сделать обрезку необратимой, если требовалась экономия дискового пространства). Так что Чед мог отсканировать фотографию своей семьи, усечь изображение, исключив всех, кроме кого-то одного, например матери, а результат использовать в домашней работе и потом, месяца через три, вернуться к той же фотографии и изменить область обрезки, включив в результат только отца, и его фотографию поместить затем в письмо. Любая другая программа сканирования заставила бы Чеда повторно сканировать изображение.

Высококлассное изменение размеров

Изменение размера изображения в большинстве графических программ означает ввод размеров в диалоговом окне. Диалог допускает высокую точность значений и позволяет растягивать изображение, меняя пропорции, однако точность требуется редко, а непропорциональное масштабирование редко когда оказывается желанным. Предлагая то, что не требуется, диалог не предлагает того, что необходимо: возможности понять, насколько большим или маленьким будет новое изображение. Инструмент масштабирования должен быть наглядным.

Наше решение: небольшой красный уголок, располагающийся в правом нижнем углу отсканированного изображения. При наведении курсора уголок едва заметно меняется в размерах, увеличивается на пару пикселов. Это я и называю «активным откликом», он показывает Малкольму, что объект поддается прямому манипулированию. Если Малкольм щелкнет мышью и потащит за красный уголок, изображение (в реальном времени) станет больше или меньше, в зависимости от направления движения уголка. Изображение всегда сохраняет правильную пропорцию. Непропорциональное масштабирование – это уже работа Магнума, который для таких целей применяет Photoshop.

Информативности инструменту масштабирования добавляют размерные линии, произрастающие из сторон изображения. Значения на линиях также изменяются в реальном времени, позволяя Малкольму при масштабировании мгновенно получать численную информацию о точных размерах изображения. Пункт меню позволяет Малкольму назначить и размерные единицы – пикселы, метрическую систему или имперскую.

Высококлассный поворот изображения

Графические программы обычно позволяют вращать отсканированное изображение. Вот три базовых применения этой функции:

• Вращение фрагментов изображений с целью изменения композиции.

• Придание нужной ориентации изображению, которое сканировалось с небольшим отклонением от вертикальной ориентации.

• Придание нужной ориентации перевернутому или перекошенному изображению.

В большинстве программ для сканеров, к которым относится и предшественник Peacock, реализован инструмент вращения, посредством которого пользователь может решать все три задачи. Мы посмотрели на всю эту мощь и сложность с точки зрения Чеда, Малкольма и Магнума, после чего решили избрать совершенно иной подход.

Первую задачу мы сразу вычеркнули из списка. Такая функция могла бы пригодиться только художнику, а художников среди наших пользователей не нашлось. Ближе всех Магнум, а он обратился бы к мощной функции поворота пакета Photoshop.

А вторая – выравнивание – не может получаться хорошо из-за ограничений технологии. Практически все растрирующие устройства, такие как видеоэкраны, сканеры, принтеры, визуализируют прямые линии, отклоненные от горизонтали или вертикали на один – два градуса, в виде жутких зазубренных линий. Такую линию не распрямить никакой компьютерной обработкой, а стандартная функция поворота в таком случае создает головокружительные оптические иллюзии. Лекарство хуже болезни. Хуже того, программный код, выполняющий поворот изображения на пару градусов, отличается большой сложностью и изощренностью. В большинстве других сканирующих продуктов эта абсурдно бесполезная функция с гордостью выпячивается – прекрасная иллюстрация того, что По Бронсон назвал слепотой, улучшающей зрение разработчиков программного обеспечения.

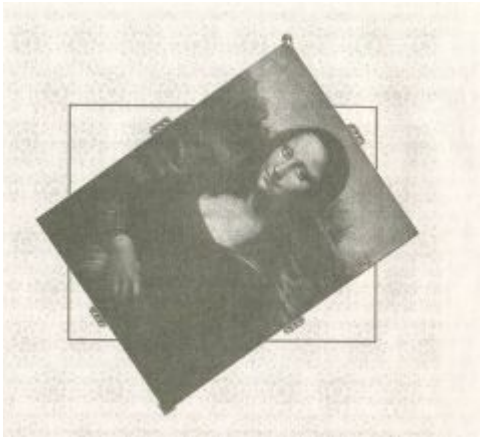
Если изображение отсканировано с одно-двухградусным отклонением, то быстрее, лучше и проще скорректировать его расположение и отсканировать повторно. Сканирующее устройство не просто поддерживает такое решение своими точными механизмами и высокой скоростью, но и с самого начала практически исключает возможность ошибки.

Третий вариант – изменение ориентации. Можно случайно отсканировать изображение вверх ногами или «уронив» его набок. Программным способом можно легко и быстро повернуть изображение на 90, 180 или 270 градусов, сориентировав его правильно.

Поэтому мы спроектировали инструмент «переориентации» вместо инструмента «вращения» и снова приложили все усилия для создания лучшего варианта. В правом

верхнем углу отсканированного изображения присутствует синий кружок, сходный с красным уголком инструмента масштабирования. Если расположить курсор над кружком, кружок слегка увеличивается в размерах, снова намекая на активность.

Если Малкольм щелкнет мышью и потащит за кружок, изображение опоясывает ярко-зеленый контур. Этот контур называется «прицелом» и подсказывает, как повернется изображение, когда Малкольм отпустит кнопку мыши. Как только кружок заходит за очередной угол изображения, зеленый прицел поворачивается в следующее из возможных положений: на угол 90, 180, 270 градусов. Малкольм заранее знает, какое влияние на изображение окажет его действие. Он отчетливо понимает, что поворот возможен лишь на фиксированные углы, а свободное вращение или коррекция результата невозможны. Все наши персонажи мгновенно понимают, как работать с инструментом.



Первоклассные результаты

По просьбе клиента мы провели пользовательское тестирование продукта и обнаружили удивительную вещь. Мы ожидали, что всем участникам тестирования очень понравится наш интерфейс, что они смогут понять его и легко им воспользоваться. Удивило то, что все как один участники высказались в том смысле, что Peacock – «самый мощный». С точки зрения количества функций это было далеко от истины. С точки же зрения доступности имеющихся возможностей продукт стал мощнее.

Когда продукт Peacock вышел наконец на рынок, персонал отдела техподдержки Logitech забеспокоился, потому что звонков с вопросами о том, как пользоваться продуктом, поступило намного меньше, чем обычно для новых продуктов.

Преодоление разрыва между устройствами и программами

С точки зрения проектировщика взаимодействия, деление между аппаратным и программным обеспечением не имеет значения – потому что оно не имеет значения для пользователя. Пользователя не интересует, какая из составляющих в производстве дороже. Таким образом, проектировщики взаимодействия способны разрешать проблемы, возникающие при создании гибридных продуктов.

В мире инженерной разработки живут разработчики аппаратной части, создающие платы электронных схем и микропроцессоры, а также разработчики программной части, создающие программный код. Хотя плоды их трудов продаются в совместных, гибридных продуктах, эти два лагеря, как правило, не сотрудничают. Иногда они даже не общаются, а просто перебрасываются готовыми модулями «через забор».

По историческим причинам разработчики аппаратной части доминируют в большинстве компаний, производящих гибридные продукты. Однако по мере

распространения аппаратного обеспечения эти устройства и разработчики этих устройств теряют свои позиции. И наоборот, все большую ценность для пользователей таких продуктов начинают представлять уникальные свойства программного обеспечения. Все это приводит к установлению тревожного перемирия в большинстве компаний, производящих гибридные продукты.

Хорошим примером такой компании является Hewlett-Packard, здесь доминируют разработчики устройств. Принтеры, производимые компанией, – продукты сказочные, с образцовой технологией, однако после двадцати лет развития ни один из моих принтеров, произведенных HP, не способен полностью интегрироваться с компьютером. Они не сообщают компьютеру, сколько бумаги осталось в лотках, или сколько порошка осталось в картриджах, или сколько заданий находится в очереди печати. Подобное бездумное пренебрежение человеческой потребностью в информации – улика, с головой выдающая компании, где доминируют разработчики устройств.

По иронии судьбы компании, выпускающие аппаратные устройства, имеют большой опыт привлечения посторонних промышленных дизайнеров для создания продуктов более полезных и желанных для потребителей. Разработчики же программ склонны создавать продукты самостоятельно. В любой компании, создающей гибридные продукты, отсутствие проектировщиков, посредничающих в интересах обеих сторон, приводит к созданию продуктов, не удовлетворяющих пользователей. Примеры главы 1 «Загадки века информации» показывают это со всей ясностью.

Гибридных продуктов становится все больше, и потребность в целеориентированном проектировании растет, потому что в смысле способов реализации оно агностично.

Продукт PalmPilot компании 3Com – хороший пример гибридного продукта, в котором проектирование позволило гладко сшить программную и аппаратную части. Достаточно коснуться экрана, чтобы машина проснулась точно в том состоянии, в каком была выключена. Мгновенная реакция устройства на действия пользователя четко показывает, что при проектировании аппаратной части были учтены потребности программной части. А вот контрпример: моя фотокамера Nikon CoolPix 900, при каждом включении которой на загрузку, при отсутствии жесткого диска, уходит семь длинных секунд. Когда устройство настолько медлительно, становится ясно, что шоу режиссировали разработчики аппаратной части.

Разумеется, в реальном мире проектирования продуктов большинство программных компаний не заходит на территорию аппаратного обеспечения. Аппаратные проектировщики поддерживают такую линию поведения даже в случаях, когда специальное устройство приобрело бы значительные преимущества в результате сотрудничества.

Однако если бюджет проекта позволяет, проектировщики могут без колебаний давать рекомендации относительно аппаратной части. Система P@ssport IFE, описанная в главе 9 «Проектирование для удовольствия», работала на выделенных компьютерах, и поставщик решения имел абсолютную власть над всеми устройствами и программами. Мои проектировщики дали некоторые рекомендации.

Продукт Elemental Drumbeat, описанный в главе 10 «Проектирование ради результата» должен был работать на любом нормальном настольном компьютере Wintel. В данном случае мои проектировщики о рекомендациях даже и не думали.

В нескольких случаях, в частности при работе над продуктом Logitech Peacock, моим проектировщикам предоставилась счастливая возможность повысить ценность аппаратного обеспечения. У каждой компании была возможность ступить на путь гибридных решений, соглашаясь со всеми потенциальными опасностями и возможностями.

Меньше значит больше

Помешанные на технике, влюбленные в контроль программисты обожают фаршировать продукты всевозможными финтифлюшками и лишними функциями, но такое

стремление противоречит фундаментальному тезису о качественном проектировании. Меньше, значит больше.

Если проектировщик взаимодействия выполнил работу очень хорошо, то пользователь вряд ли догадается о его участии. Хороший интерфейс, как обслуживание в первоклассном ресторане, не привлекает внимания. Если проектировщику взаимодействия удалось сделать что-то особенно удачное, пользователи этого даже не заметят.

«Классные» интерфейсы провозглашены целью проектирования в индустрии создания программных продуктов, и теперь артефакты взаимодействия, очевидно, отнявшие у какого-то несчастного программиста много времени и усилий, по-настоящему утомляют. Жаль, что его усилия не пошли на создание чего-то эффективного. Многие дизайнеры считают, что качественный дизайн – это классный дизайн. Часто так оно и бывает, однако *независимо от того, насколько интерфейс классный, чем его меньше, тем лучше.* Идея заключается в том, что чем меньше видит пользователь постороннего, тем лучше свою работу выполнил проектировщик. Представьте, что при просмотре фильма вам видны софиты на границах кадра, а в конце сцены вы слышите, как режиссер кричит: «Снято!» Представьте, насколько назойливы будут такие эффекты, как они разрушат очарование фильма.

Гениальный программист и дизайнер Кай Краузе (Kai Krause) прославился своими уникальными интерфейсами. Кай создал некоторые из самых мощных и интересных программ для работы с графикой. Его продукты всегда имели захватывающе красивые интерфейсы, в то же время весьма загадочные, похожие на игры. Кай не только программист, но и дизайнер, и его продукты отражают стремление дизайнера делать вещи малопонятными, как в современном искусстве – чтобы произвести впечатление. Такой подход эффективен потому, что пользователи его продуктов – такие же дизайнеры и увлеченные люди. За рамки этого мира продукты Кая пробиваются с трудом.

* * *

В программировании всегда существует бесконечное количество способов решить любую поставленную задачу. Опытные программисты знают, что в поиске вариантов, оптимальных решений, рано или поздно наталкиваешься на метод, позволяющий выбросить сотни и даже тысячи строк кода. Это происходит, когда программист совершает качественный скачок. Если он может выбросить много кода, программа становится лучше. Меньше кода – значит, меньше и сложность, меньше дефектов, меньше возможностей для некорректного взаимодействия, кроме того, упрощается сопровождение.

Проектировщики взаимодействия разделяют такой подход. Исследуя варианты, они находят точки, где можно уничтожать целые экраны или избавляться от крупных и сложных диалогов. Проектировщик знает, что каждый элемент интерфейса отягощает пользователя. Каждая кнопка и пиктограмма – это еще одна вещь, о которой пользователь должен узнать, с которой должен справиться, чтобы получить искомое. Делать больше при помощи меньшего всегда лучше.

Если проектировщик на верном пути, он *урезает* интерфейс продукта. Он вовсе не проектирует экран за экраном, наполняя их кнопками и штучками. Однажды нас посетил менеджер продукта из одной крупной компании, чтобы проконсультироваться по поводу перепроектирования продукта. В интерфейсе будет примерно десяток диалоговых окон, сказал он. Мы описали ему наш процесс, а затем дали оценку работы. Если мне не изменяет память, цифра составила $60000. «Возмутительно! – воскликнул тогда менеджер. – Это же $5000 за экран!» У меня не хватило духу сказать ему, что мы, вероятно, сократим количество диалогов до одного или двух, и тогда цена, если ее исчислять таким методом, станет гораздо выше. Он просто не понял, о чем речь. Платить за проектирование, исчисляя стоимость в экранах, все равно, что платить официанту за количество подходов к столику. Лучший официант меньше бегает, а лучший проектировщик всегда создает менее развесистые

интерфейсы.

Иногда стезя проектировщика взаимодействия становится просто невыносимой! Если, выполняя свою работу, вы делаете что-то фундаментально, абсолютно, неоспоримо верное, все тут же восклицают: «Ну разумеется! Здесь же просто нет *других* вариантов!» Такова реальность, даже если клиент много месяцев и даже лет совершенно не имел понятия, как решить свою проблему. Такова реальность, даже если наше решение приносит компании миллионы долларов. Большинство действительно новаторских прорывов *сложны в разработке и вполне очевидны задним числом.* Невероятно сложно увидеть прорыв в проектировании. Можно получить подготовку, пройти обучение, потратить многие часы на изучение задачи, но не найти ответа. Затем приходит человек со стороны, указывает на ключевую концепцию, и тут же вся головоломка складывается в полноценное изображение с естественной очевидностью, присущей колесу. Если вы закричите о своем решении во весь голос, другие ответят: «Ну разумеется, колесо круглое! Какое еще оно может быть?» Так что похвастать хорошей работой по проектированию очень и очень сложно.

Ученый-компьютерщик Алан Карп говорит: «Практически все мои патентные заявки были отвергнуты как „очевидные“».

Когда я говорю «меньше интерфейса», то не имею в виду меньше функциональности, хотя и такое случается. Я имею в виду, что пользователя не следует заставлять взаимодействовать с программой дольше, чем абсолютно необходимо для решения тои или иной задачи.

* * *

В этой главе и двух предшествующих я представил краткий рассказ о самых полезных наших инструментах проектирования. Они верой и Правдой служили для проектирования продуктов и служб – от промышленного управления до корпоративного планирования и массовых продуктов. В следующей главе я расскажу о некоторых других существующих инструментах, способствующих созданию более качественно спроектированных продуктов.

Часть V
Возвращаемся на место водителя

Глава 12
В отчаянных поисках эргономики

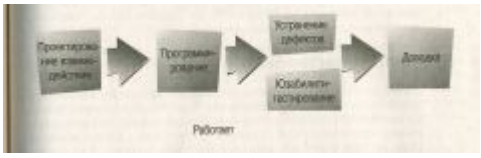
Взрывное распространение на массовом рынке продуктов, основанных на программном обеспечении, как в области универсальных компьютеров, так и специализированных устройств, преобразовало аудиторию пользователей. В прежние времена это была небольшая группа великодушных обожателей технологии. Сегодня же это огромное множество нетерпеливых, подавленных, не сведущих в технике потребителей. Наверное, каждый, кто связан с разработкой программного обеспечения, да и кто не связан, наблюдал, как в болезненном раздражении кричат пользователи, и чувствовал потребность *чем-то помочь.* Многие специалисты пошли вперед, полные решимости исправить ситуацию. У всех замечательная подготовка, у большинства отличная репутация и у многих длинные списки первоклассных клиентов. Вместе же они произвели больше тепла, чем света; их продуктам, основанным на программном обеспечении, не достает самой малости – они не являются объектом желания. Результатом стало замешательство по поводу выбора действенных способов преодоления недовольства пользователей. В этой главе я попытаюсь распутать этот клубок и показать, какую пользу может приносить та или иная специализация, и как это согласуется с целеориентированным проектированием взаимодействия.

Последовательность

Вероятно, важнейшим аспектом проектирования является последовательность событий в процессе разработки. С первых же дней разработки программного обеспечения последовательность событий была такая: программирование, устранение дефектов, доводка. Сначала программирование, устранение дефектов, доводка. Сначала программист пишет программу, затем в пошаговом режиме проходит ее в поисках непреднамеренных ошибок, внесенных при ее создании. Затем он исправляет эти ошибки, и наконец, программа готова к сдаче.

Совершенно естественно, что инженеры воспринимают любую новую дисциплину с большей готовностью, если она не нарушает привычный порядок действий. Существует метод, известный как «юзабилити-тестирование», который вырос до существенных масштабов и заключается в эмпирическом исследовании реальных взаимодействий пользователей с продуктом. Основная причина широкого признания такого тестирования в бизнесе высоких технологий в том, что оно легко встраивается в существующую последовательность. Юзабилити-тестирование по большей части связано с наличием работоспособной программы, поэтому, разумеется, необходимо ждать, пока программа не будет готова к запуску. Так юзабилити-тестирование проводится параллельно устранению дефектов, что удобно. Программистам удобна такая дополнительная форма тестирования, поскольку она не нарушает существующую последовательность действий.

Как я уже говорил, создание кода по отношению к проектированию – все равно что заливка бетонной смеси в строительстве. Независимо от профессионализма проектировщика и применяемых методов, если создание кода уже началось, воздействие проектирования окажется пренебрежительно мало. Фундаментальная предпосылка включения проектирования взаимодействия в процесс разработки программ заключена в предшествовании проектирования программированию. Очевидно, что я защищаю целеориентированное проектирование, однако любой системный процесс проектирования, предшествующий программированию, будет эффективнее любого процесса, следующего за программированием.



Проектирование взаимодействия до начала программирования означает коренные перемены в процессе разработки программного обеспечения. Естественно, такие перемены затрагивают программистов, и они видят для себя угрозу. До этого они были первыми, а

следовательно, и самыми важными. Если другая дисциплина предшествует программированию, означает ли это, что специалисты другой дисциплины более важны? Это не так, и подробности ожидают читателей в следующей главе.

Как профессионал, работавший с программным обеспечением, я занимался программированием, изобретением, тестированием, документированием, проектированием, продажей, поставкой и поддержкой. Я могу утверждать, что среди этих задач программирование – задача самая сложная и предъявляющая к исполнителю самые высокие требования. (Я говорю о *профессиональном* программировании, о создании программ, пригодных для коммерческого распространения. Известно, что сложность программы растет экспоненциально в зависимости от размера кода. В институте почти всем приходится писать небольшие, по сотне-другой строк, программы. И многие пользователи для выполнения своей работы пишут программы примерно такого же размера. Однако общий объем кода коммерческого приложения легко может превышать пятьдесят тысяч строк, поэтому сложность этих приложений выходит за пределы понимания большинства смертных.) Даже если другие специалисты об этом не догадываются, то у программистов нет сомнений, что их вклад в дело намного больше, чем, чей бы то ни было еще.

Миф о непредсказуемости рынка, рассказанный в главе 3 «Пустая трата денег», – еще одна причина, по которой последовательность «программа, тест, доводка» так прочно закрепилась в индустрии. Если мы не можем знать, чего захочет рынок, зачем тратить время на предварительное проектирование взаимодействия? Просто пишем код и выбрасываем на рынок, а там уж видно будет. Кроме того, такой подход избавляет нас от какой-либо ответственности за неудачи.

И несмотря на эти все проблемы, жизненно важно, чтобы более думающие люди изменили существующую последовательность, поместив проектирование перед программированием.

Юзабилити-тестирование

Любой процесс, основанный на наблюдении, должен играть второстепенную роль по отношению к творчеству. Программисты творят, дисциплина эргономики молчаливо передает бразды правления программистам, говоря: «Вы создадите это, а затем я протестирую и увижу, насколько хорошо вы справились.» Однако в скоростном мире высоких технологий созданный продукт немедленно выходит на рынок. Тестирование постфактум уже не может сильно повлиять на продукт.

На мой взгляд, юзабилити-методы похожи на наждачную бумагу. Если вы делаете стул, наждачная бумага может сделать его более гладким. Если вы делаете стол, наждачная бумага и его сделает более гладким. Но никакая шлифовка не позволяет превратить стол в стул. И тем не менее, мне приходится видеть, как тысячи людей из лучших побуждений прилежно шлифуют столы методами эргономики, пытаясь получить стулья.

Юзабилити-тестирование до программирования

Нет сомнений, что юзабилити-тестирование до начала программирования возможно, однако природа и ценность процесса в этом случае меняется радикально. Такого рода тестирование сравнимо с чистым исследованием, которое больше подошло бы для университетской среды. Один коллега из крупной компании, разрабатывающей программное обеспечение, провел классический юзабилити-тест, одновременно выявивший сильные и слабые места такого предварительного тестирования. Он хотел определить эффективность строки состояния внизу окна программы. Он предложил участникам выполнить безобидное задание при помощи электронной таблицы. Каждые пять минут в строке состояния появлялось сообщение: «К вашему креслу снизу приклеена банкнота в $50. Возьмите ее!» За целый день тестирования ни один из более чем десяти участников не попытался взять

купюру.

Осознать, что пользователи не обращают особого внимания на содержание популярной в среде программистов строки состояния, само по себе полезно. Однако это не проливает свет на скрытые проблемы: что есть состояние, заслуживающее внимания пользователя? Следует ли вообще отображать что-либо? В каком месте? Эти проблемы проектирования, как и раньше, остаются нерешенными.

Интеграция юзабилити-тестирования в процесс проектирования

Профессиональная литература переполнена подробными советами о том, как проводить тесты, но мало кто говорит о возможностях тестирования на стадии, когда продукт еще не существует. На практике необходимо создать и протестировать какую-то видимость программы. Обычно макет принимает форму быстро созданного прототипа или «кукольного театра», созданного из бумажных вырезок или других низкотехнологичных материалов.

Вы можете многое узнать, наблюдая за реакцией пользователей на спектакле кукольного театра, однако без предварительного проектирования может оказаться, что тестируется не совсем то, что нужно. Кроме того, присутствие человека, проводящего тестирование, оказывает серьезное влияние на такую форму тестирования: здесь любое слово, кивок и даже взгляд могут исказить результат.

Чтобы получить наиболее осмысленные результаты, необходимо провести крайне дорогостоящее сравнительное тестирование – создать две программы и каждую протестировать. И даже в этом случае вы узнаете лишь, что один вариант лучше другого. Вы не узнаете, каких высот можете достичь на практике.

Толковое юзабилити-тестирование может выявить неверные предположения проектировщика. Всегда лучше демонстрировать результаты проектирования пользователям и перепроектировать итеративно, чем вообще этого не делать. Некоторые новые технологии, такие как распознавание голоса, настолько плохо изучены, что даже результаты простейшего юзабилити-тестирования могут иметь огромную ценность.

Едва ли не самым ценным вкладом тестирования является присутствие программистов, когда они из-за полупрозрачного зеркала вынуждены наблюдать, как пользователи сражаются с их программами. Программисты испытывают шок и крайнее недоверие, они ругаются: «Вы тестируете каких-то недоумков!» Юзабилити-тестирование – меткий камень в огород упрямых разработчиков программного обеспечения, который показывает им, что проблема действительно существует. Оно может послужить той же цели и в случае с руководителями.

Перефразируя рекламу зубной пасты, можно сказать, что юзабилити-тестирование представляет собою эффективный способ предотвратить кариес в случае добросовестного следования советам профессионалов и практике целеориентированного проектирования. Главное – помнить, что другие факторы могут оказывать еще более сильное воздействие.

Многопрофильные команды

Сопротивление разработчиков программного обеспечения всему, что грозит изменить знакомую последовательность событий процесса разработки, привела к рождению многочисленных извилистых логических построений в сообществе проектировщиков. Широко обсуждается мысль о том, что проектирование должны осуществлять команды, включающие представителей многих дисциплин.

Согласно этой гипотезе, команда, включающая представителей пользователей, программистов, менеджеров, маркетологов, специалистов по юзабилити, даст лучшие результаты. По моему опыту, метод «круглого стола» не эффективен. Цели и заботы участников различаются, а участник, цели которого имеют наибольший вес, часто хуже всего приспособлен для выражения своих забот. Хуже того, программисты, в любом случае

обладающие абсолютной властью над программными артефактами, неизбежно берут на себя управление командой, обычно с заднего сиденья.

Круглый стал не дает желаемых перемен. Подход демократичный, полидисциплинарный, многокультурный, никого не оставляющий за бортом, но неспособный исправить ущербную последовательность, продолжающую отравлять взаимодействие.

Проектирующие программисты

Первыми «добровольцами» в борьбе с проблемами несведущих в технологиях пользователей стали сами программисты. Полная неприспособленность их культуры и инструментов для решения данной задачи не имела никакого значения ввиду отсутствия других кандидатов. Словно случайные свидетели, которым не повезло оказаться поблизости от места катастрофы, программисты получили задание организовать скорую помощь для интерфейсов лишь потому, что занимались родственными вещами. Программисты – ничто без своего энтузиазма, и они гордятся своей компетенцией, поэтому сложность проектирования взаимодействия привлекла их, и они приложили к решению задачи значительные усилия. Так появилась язвительная шутка: «Проектирование – то, чем программисты занимаются двадцать минут перед тем, как начать писать код».

В этой книге я неоднократно показывал, что усилия программистов были обречены с самого начала. Как говорит По Бронсон, они считают отсутствие критики комплиментом, поэтому их оценка собственной производительности невероятно высока, и многие из них продолжают держаться за роль проектировщика взаимодействия. Словно безумные короли, программисты не желают сдавать захваченную территорию, даже если оккупация неприятна, невыгодна, нежеланна и непригодна для защиты.

Если вы программируете профессионально, то вы – программист независимо от того, чему можете научить, что протестировать или спроектировать. Как нельзя быть немножко беременной, так нельзя немножко заниматься программированием.

Многие разработчики все еще не признают существования серьезной проблемы («просто пользователи должны больше учиться»), но есть и такие, кто отчетливо видит всеобщее разочарование и финансовые убытки от широко продаваемых танцующих медведей. Хорошо, что группа последних растет, как растет и желание большинства компаний-разработчиков обращаться за сторонней помощью.

В действительности большинство программистов неплохо справляется с проектированием, а многие из тех, кто не способен это делать, осознают свои недостатки и стараются не практиковать проектирование. Огромное «но»: когда программисты проектируют, их усилия практически всегда основываются на уникальной индивидуальности *хомо логикус.* Конечный результат – сложные в применении и негодные продукты, которые, как правило, нравятся только другим программистам.

Откуда вы знаете?

Многие специалисты по юзабилити считают, что невозможно понять, насколько удобно взаимодействие, пока не проведено тестирование. Поэтому они постоянно спрашивают: «Откуда вы знаете?» Однако я заметил нечто весьма любопытное. Задавая этот вопрос, они вовсе не играют в адвоката дьявола. Они спрашивают по той простой причине, что не могут опознать качественно спроектированные взаимодействия.

По меньшей мере четыре крупные компании, с которыми мне приходится работать, давно общаются с профессиональными эргономистами. Эти компании решили вложить средства в юзабилити. Они наняли профессионалов для создания лабораторий, проведения исследований, обнаружения вероятных проблемных областей и выдвижения догадок относительно улучшения эргономики. Программисты прилежно вносили изменения в

программы, *но мало что изменилось*. Разве что программистам приходилось работать намного больше. После нескольких итераций программисты просто сдались, то же самое сделали большинство менеджеров. Они поняли, что процесс очень дорогостоящий и требует больших затрат времени, однако фундаментальную задачу не решает.

Проектировщики взаимодействия полагаются на свой опыт, на подготовку и суждения, что позволяет давать точные оценки. Они используют специальные принципы, идиомы и инструменты для каждой ситуации и, наряду с этими средствами, используют еще прочие источники информации. Откуда знает рентгенолог, изучив рентгеновский снимок, что человеку требуется операция? Рентгеновские снимки настолько сложно интерпретировать, что неспециалисты просто не могут себе представить, как такое возможно. А подготовленные врачи делают это все время. Откуда судья знает, виновен ли подсудимый? Откуда инвестор знает, что настало время покупать? Эти профессионалы, возможно, делают ошибки время от времени, однако их работа основывается не на догадках.

Мне приходилось видеть, как уважаемые специалисты по юзабилити попадают «в молоко». Они разрабатывали изощренные тесты для выделения отдельных реакций пользователей на существующие программы, а затем изучали таблицы с результатами, чтобы обнаружить дефекты интерфейса. Когда их точный научный метод вскрывал проблемную область, они скатывались до дилетантских рассуждений: «Что ж, полагаю, мы можем перенести эту кнопку вот в этот диалог» или «Думаю, если мы добавим здесь кнопку, пользователю будет удобнее».

Вполне можно сказать: «Я не знаю», однако попытка угадать ответ обречена на провал. Что еще хуже – вид человека, гадающего с отсутствующим взглядом, вызывает у программистов, людей, кровно заинтересованных в результате, однозначную реакцию. Вас списывают со счетов, как шарлатана.

Руководства по стилю

Сотрудничество дизайнера Шелли Ивенсон (Shelley Evenson) и ученого Джона Райнфранка (John Rheinfrank) в исследовательском центре компании Xerox в Пало-Альто в восьмидесятые годы дало жизнь ряду важных идей в области визуальных коммуникаций. Они создали своеобразный словарь, «язык визуального проектирования» для всех фотокопировальных аппаратов Xerox: зеленый цвет обозначал оригиналы, синий принадлежности, а красный – зоны обслуживания. Схожие нетекстовые подсказки весьма полезны в интерфейсах, обладающих высоким когнитивным сопротивлением. Такие подсказки содержатся в «руководствах по стилю», содержащих примеры и предложения по использованию.

Многие разработчики программного обеспечения и их руководители, раздраженные многочисленными проблемами взаимодействия, не отказались бы иметь подобное руководство, описывающее, какой интерфейс должен быть у продукта. Многие корпорации разработали руководства по стилю интерфейсов для всех внутрикорпоративных приложений, а многие поставщики программного обеспечения предлагают подобные руководства сторонним разработчикам, занятым производством совместимых программ.

Руководства по стилю могут помочь, но они не решают проблем взаимодействия, с которыми связано целеориентированное проектирование. Эти проблемы решаются в каждом конкретном случае по-своему. Разные приложения нужны пользователям для разных целей, и каждый элемент взаимодействия в продукте должен относиться к чему-то определенному. Общий визуальный язык и последовательные органы управления способны помочь, но только их для решения проблемы будет мало.

Конфликт интересов

Потребуй Билл Гейтс публично от всех прочих компаний прекратить разработки в

области проектирования взаимодействия, эти компании просто прогнали бы его со сцены. Однако документ Microsoft, озаглавленный «Interface Style Guide» (руководство по стилям интерфейсов), требует именно этого и является одним из самых сильных конкурентных преимуществ Microsoft.

Microsoft и Apple продвигают руководства по стилям интерфейсов, превознося их мощь и пользу, и на первый взгляд может показаться, что эти компании являются самым компетентным источником подобной информации. Однако интересы владельцев платформы конфликтуют с интересами разработчиков программного обеспечения.

Оба создателя платформ применяют скрытую форму принуждения, пытаясь заставить производителей придерживаться стандарта. Независимый разработчик программного обеспечения, не следующий рекомендациям руководства по стилю, не сможет заявить о «совместимости с платформой», лишая свой продукт важного плюса с точки зрения маркетинга. Таким образом, большинство компаний, создающих пользовательские приложения, готовы следовать рекомендациям разработчика платформы.

Требуя, чтобы независимые разработчики следовали описанным принципам, разработчики платформ исподтишка подавляют инновации.

Тем временем сами разработчики платформ вольны экспериментировать и давать ход новшествам, сколько пожелают. Они могут без угрызений совести пренебрегать собственными руководствами по стилям. Разумеется, никакие другие компании не нарушают эти руководства столь же часто и откровенно, как сами Microsoft и Apple.

Я не выступаю за сожжение руководств по стилю и за хаос в интерфейсах. Просто говорю, что к руководствам следует относиться так, как сенатор относится к лоббисту, а не так, как автомобилист к полиции.

Законодатель знает, что лоббист действует из корыстных соображений, он вовсе не третья сторона, не заинтересованная в вопросе.

Фокус-группы

Многие отрасли обнаружили ценность фокус-групп, позволяющих *узнать,* что нравится и что не нравится потребителям в различных продуктах. Но какими бы полезными ни были фокус-группы для проникновения в мысли потребителей о самых разнообразных продуктах, для бизнеса программного обеспечения они могут быть источником проблемы. Самая большая сложность состоит в том, что большинство людей, включая и профессиональных пользователей программ, не представляют, что может и чего не может программное обеспечение. Поэтому если участник фокус-группы просит включить какую-то возможность, этот запрос часто носит оттенок недальновидности. Пользователь просит того, что считает вероятным, возможным, разумным. Сознательно потребовать чего-то маловероятного, невозможного, неразумного – значит просто выставить себя дураком, а люди не делают этого добровольно.

Насс и Ривз из Стэнфордского университета изучали реакцию людей на компьютеры и получили убедительные свидетельства того, что собственная оценка людей таких реакций ненадежна. Они пишут:

> «Многие распространенные методы, особенно фокус-группы, основываются на предположении, что люди способны к интроспекции в отношении [интерактивного] опыта. Мы считаем, что такое предположение часто неверно».

Ларри Кили говорит, что «пользователи отвергнут новые идеи, если вы их предложите». Поэтому фокус-группы – сомнительный инструмент для создания сколько-нибудь новаторских продуктов. В большинстве продуктов, основанных на программном обеспечении, достаточно передовых решений, чтобы фокус-группы потеряли всякий смысл.

Фокус-группы могут быть эффективными для определенных категорий продуктов, однако ошибочно будет доверять им в надежной оценке продуктов с высоким уровнем когнитивного сопротивления.

Визуальное проектирование

В книге «About Face» я показал, почему вовсе не графическая основа графического пользовательского интерфейса (ГПИ)[39] сделала его доминирующей формой взаимодействия с компьютером. Скорее, дело было в жестко ограниченном словаре новых интерфейсов. Эта ограниченность и дала ГПИ (в английском варианте GUI (Graphical User Interface)) такое преимущество перед прежними зелеными экранами. Качественный дизайн может стать хорошим вкладом в качество интерфейса, однако многие игроки отрасли все еще приписывают дизайну ценность, которой он попросту не обладает.

Я только что вернулся с конференции COMDEX в Чикаго, где судил конкурс по проектированию и созданию прикладных программ для внутрикорпоративного использования[40]. Одно из первых мест заняла программа управления продажей билетов для ежегодного съезда любителей авиации в Висконсине. Кассовый терминал – сердце системы – преднамеренно был сделан нерафическим. Небольшой текстовый экран был необыкновенно строгий, прямоугольный, эстетически примитивный. Однако программа уверенно вышла в лидеры, поскольку при проектировании пристальное внимание уделялось особым потребностям команды добровольцев, занятых продажей билетов; у этих добровольцев была ответственная, но простая работа, причем работу эту надо было выполнять быстро и с минимальной подготовкой. Графические интерфейсы замечательно подходят для информирования руководителей о положении дел в целом, но у пользователей описываемого терминала таких потребностей не было, поскольку каждый следующий клиент из очереди не имел ничего общего со всеми остальными клиентами в очереди. Видеть картину в целом в требования к программе не входило. Простого текстового экрана вполне хватило, чтобы продукт завоевал награду. Об этом уроке забывают многие профессионалы.

Одна из характерных особенностей ГПИ заключается в их способности отображать растровую графику. Можно запросто делать программные интерфейсы, насыщенные сочной графикой, как в игре Myst. Поэтому многие дизайнеры и художники с радостью наложат грим из привлекательной растровой графики на лицо вашей программы. Однако графические дизайнеры редко задумываются о сопутствующем взаимодействии.

Этот интерфейс – одна из тех бесполезных «красивостей», которые вы получаете вместе с новыми компьютерами, и он – до самой последней копейки – стоит тех денег, которые вы заплатили. Его назначение как-то связано с телефоном или приводом CD-ROM, точно не могу сказать. Интерфейс, несомненно, прекрасен, особенно если вы технофил, влюбленный во всякие примочки, однако способ работы с программой непостижим. Это пример того, что мы называем «росписью по трупам». Программисты взяли интерфейс, не подлежащий использованию из-за фундаментальных дефектов в поведенческом проектировании, и завернули его в привлекательную обложку.

[39] В английском варианте GUI (Graphical User Interface). – *Примеч. науч. ред.*

[40] Конкурс проводится уже семь лет и называется Windows World Open. Спонсируется компаниями Microsoft, Computer World и Ziff-Davis Events.

Поставщики устройств, похоже, особенно увлечены таким подходом, ведь программа шла в комплекте с моим новым компьютером. Подозреваю, дело в том, что интерфейс метафорически следует представлениям о классном аппаратном обеспечении.

Мы часто видим красивые продукты, эстетика которых великолепна, а функциональность или способы взаимодействия неадекватны назначению. Причина не в том, что продукт не проектировался, а в том, что он проектировался дизайнером, а не проектировщиком взаимодействия, имеющим инструменты для борьбы с когнитивным сопротивлением.

Промышленный дизайн

Другая область, к представителям которой обращаются за экспертизой, это промышленный дизайн. Эта хорошо развитое и сформировавшееся ремесло создания трехмерных объектов, приспособленных для вашего взгляда, вашего тела и особенно для ваших рук. В целом промышленные дизайнеры достигают отличных результатов, и упрекать их можно разве что в недостатках, но не в проступках. Они обучаются созданию кнопок, регуляторов и органов управления, которые легко увидеть, нащупать, применить. Однако они не подготовлены специально ни к разрешению проблем когнитивного сопротивления, ни к сотрудничеству с разработчиками программного обеспечения. Кнопки мгновенно опознаются как таковые (для примера возьмите систему дистанционного замка из главы 2 «Когнитивное сопротивление»), даже на ощупь. Их физическое назначение интуитивно понятно, однако логическое назначение (метаназначение) остается неясным, как прежде.

Каждый из пяти пультов на моем кофейном столике в отдельности сделан хорошо, но все вместе они делают мой домашний развлекательный комплекс практически бесполезным. Обладая привлекательным внешним видом, эти пульты бесполезны, если требуется переключить канал или заглушить звук в темноте. Дизайнеры, создавшие эти пульты, удовлетворили требования разработчиков развлекательных систем, но не удовлетворили потребности пользователя в качественном взаимодействии.

Легко понять, почему менеджеры продуктов путают промышленный дизайн с проектированием взаимодействия. Промышленные дизайнеры тоже имеют дело с интерфейсом между людьми и продуктами высоких технологий. Они тоже облегчают людям применение высокотехнологичных штуковин. Кнопки можно легко найти и нажать, но это не означает, что пользователь поймет, какую именно кнопку следует нажимать. Это проблема когнитивного сопротивления, а не промышленного дизайна.

Классная новая технология

И последний претендент на трон проектирования взаимодействия – сама технология. Microsoft, в частности, усиленно навязывает эту ложную панацею. Например, Microsoft утверждает, что интерфейсы упростятся, как только разовьются технологии распознавания голоса и рукописного ввода. Уверен, что это глупость. Каждая новая технология всего лишь

добавляет возможностей для приведения пользователей в отчаяние – при помощи систем более быстрых и более мощных.

Ключ к более качественному взаимодействию кроется в уменьшении неопределенности между компьютерами и пользователями. Обработка естественных языков никогда не даст такого эффекта, поскольку значения слов в человеческой речи весьма расплывчаты. Наше общение столь часто основано на нюансах, жестах и интонациях, что появившиеся системы распознавания слов еще очень долго не смогут (если вообще смогут) научить компьютеры интерпретировать значения наших фраз.

Технология распознавания голоса определенно принесет пользу многим продуктам. Но, полагаю, было бы излишне оптимистично надеяться, что какая-то новая технология сможет просто спасти нас, сделать то, что до сих пор не было сделано. Технология требует проектирования взаимодействия для создания законченных решений для пользователей, независимо от того, какое сочетание технологий используется.

Итерации

Расхожая истина о разработке программного обеспечения гласит, что добиться качественного взаимодействия можно только поэтапным улучшением. Приверженность юзабилити-тестированию во многих крупных компаниях, в частности в Microsoft, привела к распространению этой идеи. Конечно, итерации – важный элемент качественного проектирования: продолжаем работать, пока не получим правильное решение. Однако многие разработчики поняли идею иначе: плюем на проектирование и просто пересчитываем в темноте все кочки, какие есть на дороге.

В 1986 году компания Microsoft поторопилась на рынок с первой версией Windows, которая была столь смехотворна, что заслуженно стала предметом шуток. Шесть месяцев спустя Microsoft выпустила версию 1.03 и исправила некоторые дефекты. Годом позже Microsoft выпустила версию 1.1, а затем версию 2.0[41]. На каждом этапе развития продукта разработчики пытались разрешить проблемы, созданные в предыдущей версии. Наконец, четыре года спустя после выпуска первой версии, Microsoft представила Windows 3.0, и все перестали смеяться. Мало какие компании в этой индустрии имеют такое упорство и финансовые возможности, позволяющие выдержать четыре года публичного унижения и добиться, наконец, приемлемого результата. При этом все наблюдают, как лидер де-факто слепо спотыкается практически до победного конца, после чего делают очевидный вывод о том, что именно так и нужно действовать.

Однако создание всех этих промежуточных версий дорого обошлось компании. Если бы Microsoft достигла уровня качества Windows 3.0, пропустив пару промежуточных версий, она сэкономила бы миллионы на разработке и поддержке, при этом заработав дополнительные миллионы на продажах – на более раннем этапе жизни продукта. Позиция, защищающая неизбежность создания многочисленных версий продукта, – весьма дорогостоящее убийство здравого смысла.

Стратегия Microsoft основана на изморе. Говоря военным языком, враг может быть равен вам в умении сражаться (и даже превосходить вас), но, обладая огромным численным перевесом, вы можете просто обмениваться жертвами, пока у противника не закончатся войсковые соединения; в терминах производства программного обеспечения это означает: выбрасывайте на рынок некачественный продукт, пусть даже это танцующий медведь, а затем слушайте жалобы и стоны своих клиентов. Доводите до ума то, что им не нравится, и выпускайте обновленную версию. После трех или четырех версий открытые очаги болезней

[41] В нумерации версий продуктов Microsoft совершенно нет логики. Windows 3.0 предшествовали по меньшей мере четыре значительных издания системы. Очевидно, что Windows 3.1 – радикально иной и улучшенной версии, содержащей массу серьезных изменений, следовало дать название Windows 4.0. Уверен, что маркетологи Microsoft дали этой версии номер 3.1 потому, что не хотели терять рыночную нишу, уже завоеванную «третьей версией».

пользователей потухнут, а качество достигнет какого-то приемлемого минимума, поддерживаемого широкой функциональностью, после чего расти уже не будет. Итеративный подход не позволяет создавать замечательные продукты.

Стратегия измора не просто дорого обходится и заставляет тратить уйму времени, она омерзительна, потому что негуманна по отношению к пользователям компьютерных технологий. К несчастью, эта стратегия неплохо служит Microsoft. Компания не устает выпускать сырые, непродуманные, плохо сконструированные и спроектированные продукты на потеху индустрии и осмеяние наблюдателей, как пристрастных, так и беспристрастных. Однако пока специалисты отпускают язвительные замечания, Microsoft продолжает поддерживать свои первые попытки вторыми, третьими, четвертыми, пятыми, наконец, одиннадцатыми версиями. Такие продукты, как Windows, ActiveX, Word, Access, Windows NT и многие другие, в конечном итоге стали гигантами соответствующих рыночных ниш.

Стратегия измора эффективна, только если применяется компаниями, обладающими железобетонным именем, кучей времени, выдержкой игрока в покер и неисчерпаемыми финансами. До сих пор ни один участник компьютерной индустрии не проявил все эти качества на уровне, соответствующем уровню Microsoft.

Впечатляющий коммерческий успех Microsoft плох тем, что многие не столь крупные компании пытаются повторить ее успех, действуя такими же методами. В долгосрочной перспективе подобная стратегия часто заканчивается плачевно, что показал пример Netscape, однако она продолжает традицию надругательства над конечными пользователями.

Игрока, применяющего стратегию истощения, вполне можно победить, но только не подобным подходом. В конце концов, независимо от уровня вашей компании, денег у Microsoft больше. Следует наносить массированные удары по слабому месту Microsoft, а именно в области процесса разработки, ставящего программирование перед проектированием взаимодействия. Microsoft дважды ущербна в том смысле, что многие люди в компании занимают должность «проектировщика» и выполняют функции, связанные с проектированием. Как показывают выдержки из книги Фреда Муди (см. главу 8 «Отмирающая культура»), культура Microsoft уже привязалась к неэффективному «проектированию постфактум». Любая компания, готовая заниматься *реальным* проектированием взаимодействия, сможет побить Microsoft.

Глава 13
Управляемый процесс

Полагаю, большинство менеджеров, руководящих созданием продукции, основанной на программном обеспечении, в действительности не имеют четкого представления о том, как распознать лучший и наиболее успешный продукт и как создавать такие продукты. Как следствие, они руководствуются своим страхом, а значит, шутят с огнем. Они действуют ловко, но не владеют ситуацией и рискуют обжечься, отвлекшись даже на секунду. В этой главе я расскажу о дилемме, перед которой оказывается технический руководитель, и покажу, как укротить огонь при помощи проектирования.

Кто на самом деле самый влиятельный?

Как понять, чьему совету следовать, а чей можно игнорировать? Мне приходилось встречать руководителей, поведение которых не сильно отличалось от поведения собак на перекрестке. На забитом автомобилям и перекрестке эти собаки яростно лают, пытаясь бежать сразу во всех направлениях. Руководство говорит: «Пусть будет похоже на Outlook 98». Маркетологи говорят: «Хотя бы на уровне конкурентов». Отдел продаж говорит: «Этот клиент требует вот такую функцию». Программисты говорят: «Необходимо сохранять совместимость – делаем, как в последней версии». Кому верить?

Руководители разработки продукта стараются соглашаться со всеми участниками процесса. Влияние программистов несоразмерно, потому что они владеют кодом, а потому их целям отдается обычно наибольший приоритет. Но есть и еще одна группа, потребности которой, казалось бы, имеют высший приоритет. Это клиенты. В конце концов, сколько бы участников ни вносили свои предложения, *только клиент держит в руках чек.* Ни один деловой человек не оставит это без внимания!

Смертельная спираль: на поводу у клиента

Приняв этот чек, вы превращаетесь в компанию, «ведомую клиентами». Идея звучит симпатично и широко реализуется, однако является ошибочной. Вы начинаете откровенно играть с огнем. В восьмидесятые годы IBM очень гордилась статусом компании, ведомой клиентами. Тогда IBM владела практически всем компьютерным бизнесом, в масштабах более серьезных, чем сегодня Microsoft, однако сейчас, оставаясь крупной компанией, IBM – лишь одна из многих, но никак не лидер.

Обычно новая компания основывает свой первый продукт на каком-то технологическом новшестве. Этот первый продукт проектируется, исходя из внутренних представлений о том, как все следует делать. На этом этапе клиенты компании еще не проявляют большой заинтересованности, поэтому их советы бессвязны. Но как только продукт появляется на рынке, заинтересованность клиентов начинает увеличиваться, поскольку они вкладывают в продукт свое время и энергию. Естественно, что от них приходят запросы по изменениям и добавлениям.

Существует большая разница между тем, чтобы *выслушивать* клиентов, и тем, чтобы за ними *следовать*. Выслушивать разумно. Подразумевается, что вы накладываете свой собственный фильтр на услышанное. Следовать требованиям клиента – плохо. То есть просто делать все то, что говорит клиент. Уже не вы играете с огнем, огонь играет с вами.

Как только производитель продукта позволяет своим клиентам диктовать, какие возможности будут в продукте, происходит очень серьезная, но практически незаметная перемена. Компания перестает быть производителем *продуктов,* изобретающим вещи, которые можно продавать клиентам, и становится *обслуживающей* компанией, выполняющей работу по запросам клиентов. Каждый в компании ощущает эту тонкую перемену и реагирует совершенно верно, выступая за интересы клиентов сильнее, чем за какие бы то ни было другие.

Сегодня многие компании, создающие корпоративное программное обеспечение, такие как Oracle и SAP, в начале девяностых пережившие взрывной рост в ходе замены старых архитектур новыми клиент-серверными, заново испытывают кошмар клиентского управления, идя по стопам IBM. Представив новую технологию, эти так называемые ERP-компании (Enterprise Resource Planning, планирование ресурсов предприятия) стали прислушиваться к своим клиентам. Они начали добавлять возможности, затребованные клиентами, не соотнося их с более масштабными долговременными планами.

Мне приходилось слышать от руководителей, что никакие изменения не вносятся в их продукты, пока этого не потребует клиент. Каждый клиент ведет дела немного иначе, чем все остальные, и каждый требует, чтобы ERP-компания внесла изменения и добавила возможности, соответствующие именно его способам ведения дел.

В ложно понятом стремлении быть полезной, компания, слепо идущая на поводу у клиентов, вынуждена делать уступки.

Компания одна, а клиентов у нее будут десятки и сотни. Если реагировать на всех (или хотя бы на самых крупных), кто возьмет на себя труд урегулирования этих противоречивых требований?

Многие из знакомых мне руководителей в области высоких технологий имеют опыт инженерной разработки и часто раньше работали программистами. Можно точно сказать, что они занимают свои посты потому, что очень хорошо знают программистов и

испытывают к ним симпатию. Как показано в главах 7 «Homo Logicus» и 8 «Отмирающая культура», программисты в поисках ответа обращаются к функциям и возможностям. Когда клиент приходит с перечнем возможностей в одной руке и чеком в другой, технический руководитель не способен сопротивляться такому сочетанию. Вот еще одна причина, по которой столь многие организации, создающие продукты, ведут с заказчиками торги за функции, – чтобы управлять сроками разработки. Они играют с огнем, а управление, основанное на жестких сроках сдачи, гарантирует, что скорость этой игры будет только нарастать.

Концептуальная целостность – важное достоинство

Приняв чек, вы отдаете бразды правления клиенту. У клиента могут быть деньги, однако клиент не обладает двумя важнейшими качествами: 1) он не заботится о ваших интересах и 2) не знает, как проектировать ваш продукт.

Продукты, создающиеся под дудку клиентов, не обладают ясным замыслом. Такие продукты не имеют качества, которое идеолог программного обеспечения Фредерик Брукс называет «концептуальной целостностью, всепроникающим видением программы», которое, по его мнению, и есть главный ингредиент успеха. В отсутствие концептуальной целостности могут случиться две вещи: клиенты начинают контролировать проектирование вашего продукта, а вы перестаете это делать. Неважно, насколько у клиента благие намерения, он не обладает способностью воспринимать ваш продукт, как единую концептуально целостную сущность. Четкое видение ситуации – это одно из главных достоинств, и большинство компаний с трудом могут сфокусироваться на собственном бизнесе, что уж говорить о вашем. Даже выкрикивая противоречивые приказы, клиенты ожидают, что *вы* самостоятельно выберете правильные.

Если продукт разрабатывается под дудку покупателя, он мутирует от версии к версии, вместо того, чтобы расти упорядоченно. В конечном итоге продукт состоит из несовместимых фрагментов и случайно подобранных возможностей, продукт становится, по выражению Джона Зикера (John Zicker), «собачьим завтраком». Каждому клиенту приходится продираться через продукт, находя возможности, которые ему нравятся, избегая возможностей, которые ему не подходят, и все без исключения клиенты находят, что пользоваться продуктом становится все сложнее и сложнее с каждой новой версией. Некоторые известные компании создают продукты настолько сложные, что даже решение простейших задач требует многомесячной подготовки. Появляются целые бизнес-направления для установки, настройки, сопровождения, обучения работе с этими монстрами. Клиенты могут покупать собачьи завтраки, но вряд ли кто-то сможет влюбиться в данный продукт. Такой продукт не хотят покупать, что, как я показывал в главе 5 «Нелояльность клиентов», делает вас весьма уязвимой мишенью конкурентов.

Фаустова сделка

Такой переход от компании, ведомой определенным видением, к компании, идущей на поводу у клиентов, можно рассматривать как превращение производственной компании в обслуживающую. В замечательной книге «Managing the Professional Service Firm» (Управление компанией профессиональных услуг) Дэвид Майстер[42] рассуждает о проблеме следования за клиентами в контексте иных услуг, услуг по консультированию. Разумеется, сфера услуг значительно отличается, и Дэвид применяет иную терминологию. Он противопоставляет продажу умения *решать проблемы* и продажу прошлого *опыта*. Эти варианты он называет, соответственно, «мозг» и «седая голова» (опыт).

Продавать мозг сложно. Человек, нанимающий вас в качестве мозга, должен вам очень

[42] David Maister «Managing the Professional Service Firm», 1997, Free Press, New York.

сильно доверять, поскольку ожидает, что вы проделаете нечто, в чем ваша компетенция еще не доказана. Продавать опыт проще. Потенциальный клиент может увидеть, что вы уже прежде решали подобные проблемы, а значит, справитесь и с его трудностями.

Большинство начинающих консультантов продают «мозг» своим коллегам, то есть людям, доверительные отношения с которыми уже установлены. Как только консультант решает проблему клиента, то начинает приобретать опыт, что приводит к наплыву клиентов. Новые клиенты будут все менее знакомыми, изначально доверяющими консультанту все меньше. Поэтому они будут предлагать консультанту работу, требующую знаний и опыта. В конце концов, нового клиента привлек именно опыт; именно такое задание клиент склонен дать непроверенному исполнителю.

С приобретением репутации растет число клиентов, привлеченных опытом консультанта, и консультант обнаруживает, что опыт позволяет зарабатывать больше и легче. В конце концов, он ведь делает все ту же работу, что и раньше.

По мере того, как консультант все меньше продает свой интеллект и все больше – опыт, исчезают именно те качества, которые и делали консультанта ценным. И он начинает отставать. Услуга, которую он теперь предлагает, уже связана не с блестящим решением проблем, а лишь с приземленным решением задач. Популярность консультанта снижается, и его собственные клиенты начинают давать ему все более унизительные задачи. Они начинают посматривать в сторону других консультантов, идущих далеко впереди, на тех, кто продает интеллект.

Это та же смертельная спираль, в которую попадает тот, кто идет на поводу у клиента, только теперь это компания, предоставляющая услуги.

Сделаем выводы. Идущий на поводу у клиента получает легкие деньги сразу, но перестает расти и ставит крест на собственных перспективах. Он отказывается от своей роли лидера.

В этой игре тайно участвуют все. Клиентам она весьма удобна. Новые клиенты приходят и говорят: «Добавьте вот эту возможность в свой продукт, и я его куплю». Это проверка, позволяющая узнать, предоставляете ли вы услуги нужного характера. Отдел продаж вкладывает огромные усилия в такие крупные продажи, и кажется, что добавление одной маленькой возможности – небольшая цена за установление отношений с новым покупателем. Прибыль манит.

Решение, предложенное Майстером, очевидно: больше участвовать в проектах, задействующих мозг. В контексте услуг необходимо убеждать существующих клиентов, привлеченных вашим опытом, давать больше задач для мозга, и Майстер подробно описывает, как это делать. Это означает, пишет он, что придется отказываться от легких заработков на опыте в пользу более тяжелых и не столь прибыльных проектов для мозгов. Переводя решение Майстера в область разработки продуктов, мы обнаруживаем, что все запросы клиентов – это задачи для опыта, тогда как все задачи для мозга появляются внутри компании. Иными словами, *ваша* задача как руководителя разработки продукта – удержаться на передней линии, избегая при этом смертельной спирали, в которую вас увлекают клиенты. Ответы следует искать внутри себя, делать то, что вы делали, когда только начинали бизнес.

И это означает более тщательное прогнозирование, принятие ответственности, затраты времени, удержание управления.

Прогнозирование

Чтобы сохранить конкурентные преимущества, необходимо рассматривать краткосрочные прибыли в перспективе. Убедитесь, что ваши люди понимают, что краткосрочное планирование равносильно закладке тикающей бомбы в собственное сердце. Краткосрочной перспективы следует избегать, несмотря на возможные расходы.

Долгосрочное прогнозирование означает отказ от некоторых очень привлекательных сделок. Делать это сложно, но необходимо для выживания в будущем. По моему опыту,

такие сделки редко когда действительно теряются. Если вы обладаете достаточной уверенностью, чтобы отвернуться от клиента, предлагающего деньги, клиент, вероятнее всего, больше будет вам доверять и подвергнет переоценке свои запросы. Однако такие поступки требуют воли.

Принятие ответственности

Необходимо как можно раньше обрести баланс. Нельзя руководствоваться, например, такими соображениями: «Пару лет всего с ближним прицелом поработаю, а потом переключусь на долгосрочное прогнозирование». Необходимо найти баланс в первый же день. Отложить можно ближний прицел, а вот долгосрочное планирование откладывать нельзя.

Здесь речь идет о корпоративной культуре, и в уже принятое такой культурой краткосрочное планирование очень тяжело внести изменения. Рискованно не идти к прибыльной пропасти, каковой является следование запросам клиентов, – вы вызовете огонь на себя. Черпайте мужество в том, что поступаете верно.

Затраты времени

Многие высокотехнологические компании считают за правило выпускать новые версии программ каждый год. Некоторые выпускают версии еще чаще. Это означает, что основная масса программистов работает в годичных циклах, и любая работа должна пройти путь от идеи, через проектирование, через программирование, через тестирование, к рыночному воплощению в пределах этого года. Для каких-либо новаций в проектировании этого времени недостаточно, поэтому большинство компаний стараются совместить проектирование с программированием. Как я уже подробно рассказывал, если скрестить проектирование и программирование, получается просто программирование.

Удержание управления

Самое главное для руководителя разработкой – отнять контроль над процессом у бушующего огня. Необходимо укротить этого зверя, приняв сопутствующие травмы, как неизбежность. Если выживете, то сможете начать перестраивать процесс таким образом, чтобы обрести баланс между интеллектом и опытом и сохранить в будущем это преимущества.

Поиск основы

Большинство компаний проводят очень аккуратное планирование и анализ финансовых и операционных сторон функционирования бизнеса. Что же касается продукта, они считают, что перечень возможностей – достаточно качественное планирование, хотя оно совершенно таковым не является. Разработка программного обеспечения – процесс слишком сложный, слишком дорогостоящий, слишком тяжелый, чтобы бросаться в него без тщательного планирования и прогнозирования. В контексте разработки программного обеспечения «тщательное планирование» может означать лишь проектирование взаимодействия, которым, как мы установили, довольно часто пренебрегают.

Одной из побочных выгод целеориентированного проектирования является набор персонажей: перечень конкретных типов пользователей. Этот документ оказывается значительным подспорьем при необходимости реагировать на запросы пользователей. Прежде всего, определите, какому персонажу может послужить новая возможность, а затем – является ли этот персонаж одним из ведущих. Если так, можете отнестись к запросу всерьез. Если нет, добавление этой функции приведет к отставанию, независимо от того, сколько

денег вы получите. Если к вам в офис придет клиент и предложит $100000, чтобы вы выбросили свою систему бухгалтерского учета или подожгли ящики с бумагами, сделаете ли вы это?

Семь раз отмерь

Если компания идет на поводу у клиентов, это четкий симптом того, что руководители разработки продуктов верят в миф о непредсказуемом рынке. Но они не знают, хороша или плоха возможность, необходима она или же не нужна. Они просто передают бразды правления клиентам: «а почему бы и нет?» *Сами они* определенно этого не знают. Если клиент говорит: «Добавьте в функции разводной гаечный ключ для левшей», руководитель считает, что клиенту, наверное, что-то известно, чего он сам не знает. Руководитель верит, что именно эта возможность может волшебным образом принести продукту ошеломительный успех.

Обратная сторона медали – руководитель не имеет представления и о том, какие возможности следует убрать. Когда внешние силы ограничивают расписание, руководитель должен пожертвовать какими-то возможностями, однако понятия не имеет, какие возможности жизненно необходимы, а какие – попросту «подливка».

Разрешить непроектировщикам выкидывать возможности? Все равно, что разрешить пассажиру перерезать провода в самолете. Такая вивисекция делается наугад или основывается на каких-то не имеющих к делу отношения качествах вроде цвета изоляции или расстояния от кресла этого пассажира. Могут быть перерезаны и нужные провода. Можно просто отключить свет для места 22A, а можно вывести из строя двигатели. Проектировщики же режут возможности так, как режет создатель самолета: не затрагивая те провода, которые нужны для полета.

Производство фильмов

Производство фильмов – занятие непомерно дорогостоящее, как и создание программного обеспечения. Голливуд снимает фильмы дольше, чем мы производим программы в Кремниевой Долине, и нам есть чему поучиться у них. Действительно дорогостоящая часть – это непосредственная съемка фильма. Все эти камеры, сцены, техники, актеры ежедневно съедают многие тысячи долларов. Хорошие фильмопроизводители четко контролируют эту стадию и заранее планируют все подробнейшим образом. Тратя время и деньги на создание подробных раскадровок и съемочных расписаний, они экономят кучу времени и денег во время съемок.

Процесс создания фильма можно разбить на три крупных стадии: подготовка, производство, доводка. На стадии подготовки появляется и дорабатывается сценарий, выполняются работы по дизайну, нанимаются актеры и персонал для съемок, находятся инвесторы. На стадии производства зажигаются софиты, включаются камеры, режиссеры выкрикивают указания, актеры играют роли. На стадии доводки происходит монтаж фильма, запись звукового сопровождения и проработка маркетинговой кампании. Эти стадии достаточно хорошо соответствуют стадиям разработки программного обеспечения.

На стадии подготовки руководители занимаются проектированием взаимодействия для

продукта, нанимают программистов и находят инвесторов. На стадии производства зажигаются мониторы, компиляторы включаются в работу, руководители выкрикивают указания, а программисты пишут код. На стадии доводки отлаживается код, пишется документация, прорабатывается маркетинговая компания.

Важная особенность этого трехстадийного процесса: подготовка позволяет *минимизировать длительность стадии производства.* Зеленый свет началу съемок стоит невероятно дорого, поэтому экономия нескольких дней съемки может возместить многие недели работы на стадиях подготовки и доводки. Подготовка к съемке занимает год и более, интенсивные съемки – пару месяцев, а доводка – еще многие и многие месяцы.

Более того, по мере роста технической сложности фильмов (что получится, если скрестить фильм с компьютером?) все меньше остается производственной работы, которую можно выполнить без тщательного предварительного планирования. Если планируется дуэль кинозвезды и созданного на компьютере пришельца на лазерных мечах, эта дуэль будет сниматься на фоне синего экрана и в действительности не состоится, поэтому все действия актера вплоть до мельчайших движений и направления взгляда следует спланировать.

Создатели фильмов знают, что у них будет лишь один шанс сделать все верно, поэтому никогда не забывают о стадии подготовки. В мире компьютерной разработки многие руководители считают, что смогут все починить в следующей версии, поэтому давление в области планирования снижается. Это предположение обходится ужасающе дорого.

Сегодня создавать программы так же сложно, как фильмы, однако процесс разработки по преимуществу игнорирует этот факт. Мне встречаются в основном команды разработчиков, которые тратят несколько дней или недель (максимум) на планирование и проектирование, затем от 6 до 18 месяцев – на программирование, затем всего пару месяцев занимаются отладкой, тестированием и документированием. Подозреваю, нам многому следует научиться у киноиндустрии. Если бы мы больше времени тратили на стадию подготовки, на проектирование, то смогли бы сэкономить массу дорогостоящего времени программистов.

Стадия подготовки в киноиндустрии – самая дешевая. Затраты на создание раскадровки дорогостоящей погони со взрывами и специальными эффектами не очень велики. Чтобы внести радикальные изменения, достаточно ластика, карандаша и какого-то времени. Тщательная детализация на бумаге экономит миллионы в тот момент, когда включается камера, а в автомобили набиваются каскадеры и взрывчатые вещества. Подготовка – это инвестиции во время, которое экономит наличные средства и повышает вероятность успеха.

Если режиссер передумал и хочет взорвать вертолет вместо поезда, то на стадии подготовки сделать это просто и обойдется недорого. Подобные изменения во время съемки практически самоубийственны. Создатели фильмов это знают, а потому не торопятся во время подготовки, а во время производства следуют этим планам.

Для уже существующих продуктов цикл искажается еще больше. Ежегодное обновление возможностей для уже вышедшего на рынок продукта может вообще не оставлять времени на предварительное проектирование. Руководитель разработки продукта просто ведет перечень возможностей, запрошенных клиентами, и каждый год без лишних церемоний вручает этот список программистам.

Как и в случае производства фильмов, стадия тщательного проектирования продукта в процессе разработки может дать огромные преимущества. Создавая на бумаге подробный план для продукта, мы исключаем многочисленные неопределенности в программировании, а также существенно снижаем риски, обычно сопутствующие выпуску продукции, основанной на программном обеспечении.

Хорошая сделка

Руководство должно взять на себя ответственность и включать проектирование в процесс до того, как начато программирование. Если проводить аналогии, проектирование

взаимодействия – это архитектура, а не дизайн интерьеров. Проектирование взаимодействия определяет, куда будет залит бетон фундамента, точно так же, как и самый подходящий материал для занавесок или портьер на окна. Проектировщики взаимодействия должны получить моральные полномочия диктовать форму и конституцию продукта программистам. Это приведет к серьезным культурным сдвигам, но после таких перемен программисты станут счастливее, а вы получите выгоды от значительно более совершенного продукта.

В обмен на такую власть сообщество проектировщиков взаимодействия также должно принять определенную ответственность. Во-первых, проектировщики должны войти в число лиц, кровно заинтересованных в исходе игры. Они должны сойти с боковой линии, откуда сейчас дают советы программистам, разрешая тем принять полную ответственность за успех продуктов. Недостаточно просто иметь правильные идеи. Необходимо добиться применения этих идей на практике, и единственный способ это сделать состоит в том, чтобы проектировщики взаимодействия встали ближе к точке риска. Программисты приближаются к этой точке с каждой строкой кода.

Кроме того, проектировщики взаимодействий должны фиксировать свои предложения в письменной форме.

Документируйте замысел, чтобы дать ему жизнь

Один из действительно жестких уроков, которые мне пришлось усвоить за прошедшие годы, таков. Хорошее и даже отличное проектирование бессмысленно, если не воплощается в жизнь. И оно никогда не воплощается в жизнь, не будучи описанным подробно, точно и со всеми деталями, причем в терминах, имеющих смысл для программистов. Описание должно быть письменным, включать исчерпывающие подробности, сопутствующие свидетельства и примеры. Оно должно быть напечатано и переплетено во множестве экземпляров. Его следует лично представить команде разработчиков. Вице-президент разработки в этот момент должен присутствовать, кивать и улыбаться. Еще лучше, если это будет президент компании.

Необходимо, чтобы проектировщики писали, кадрировали, анимировали, прототипировали свои решения настолько полно и подробно, чтобы программисты могли считать эти решения чертежами и писать на их основе код. Необходимо описывать достаточное количество ситуаций, давая разработчикам уверенность в том, что решение достаточно надежно, чтобы дожить до реализации.

Проектирование в письменной форме подобно плану сражения. Каждый знает свою роль, обладает критической и своевременной информацией. Все могут действовать сообща и гармонично, создавая продукт, ориентированный на конкретного пользователя.

Программисты прибегают к довольно действенной «пассивно-агрессивной» тактике. Не вступая в конфронтацию, призванную разрешить вопрос, они избегают внимания к нему и втихую предпринимают – или не предпринимают – определенное действие. Как пассажир, который направляет каноэ, исподтишка наклоняясь то влево, то вправо. Одна из моих любимых бизнес-аксиом – «если этого нет на бумаге, значит, это не существует» – в мире проектирования программного обеспечения верна, как нигде. Если что-то не записано, более чем вероятно, что это будет неправильно истолковано или опущено. Дело в том, что мотивация программистов очень сильно расходится с мотивацией пользователей. Недостаточно исключить диалоговое окно из описания, проектировщик обязан явным образом указать, где программист не должен по собственной инициативе добавлять лишнее окно. Программистам нравятся диалоговые окна. Они считают, что оказывают пользователю услугу, добавляя в свободные минуты пару дополнительных диалоговых окон. А пользователи эти окна ненавидят, потому что диалоги их изматывают и снижают производительность.

Проектировщик взаимодействия, как архитектор, сдает набор чертежей, регламентирующий продукт, который должен быть создан. Но, несмотря на сходство, между

чертежами и проектировочными документами есть и большие различия. Чертежи обладают большей емкостью. Единственная линия чертежа может обозначать стену из ста тысяч кирпичей. Когда речь идет о взаимодействии, емкость улетучивается. На описание поведения ста страниц кода может уйти сто страниц документа. С минимальной долей шутки я заявляю, что *достаточно детализированная спецификация неотличима от кода, эту спецификацию реализующего.* В идеальном мире разработчики будут предоставлять проектировщикам целый год для проектирования, а затем программистам три месяца на создание кода. В сегодняшнем мире все наоборот.

Из этого следует, что проектировочный документ должен, это неизбежно, опускать некоторые моменты. Проектировщик должен иметь не только чувство качественного проектирования, но и понимать, что именно сейчас важно. Проектировщик взаимодействия должен решать, какие части программы необходимо спроектировать, а какие можно оставить на откуп программистам.

Все мои проектировочные документы построены по принципу спирали, как газеты. Заголовок новости содержит всю значимую информацию. Первый параграф повторяет новость более подробно. Следующие три параграфа повторяют новость еще раз, еще более подробно. В оставшейся части статьи, которая может занять несколько колонок, содержится вся история в мельчайших подробностях. Все это позволяет читателю получить необходимое, не разгребая ненужные детали.

Проектирование влияет на код

Многие ошибочно полагают, что проектировщики взаимодействия делают дизайн интерфейса пользователя. Без дизайна интерфейса, несомненно, не обойтись, однако в процессе проектирования интерфейс играет второстепенную роль, примерно как упаковка продукта. Дизайн интерфейса должен выполняться после того, как определено назначение и поведение интерактивного продукта. Некачественный продукт в красивой и красочной упаковке – продукт по-прежнему некачественный.

Специалистов по дизайну интерфейса обычно приглашают в момент, когда работа над продуктом завершена или близится к завершению, возможности для проектирования в основном уже упущены, и здесь усилия проектировщика – не важно, насколько героические – имеют ограниченное действие.

Проектирование взаимодействия может радикальным образом изменить процесс реализации, хотя это не означает, что проектирование затрагивает только вопросы реализации. К примеру, программисты могут ожидать, что проектировщик опишет внешний вид важного диалогового окна. А проектировщик может пожелать заменить это окно чем-то другим, например панелью инструментов. При этом он не интересуется программной реализацией диалоговых окон и панелей инструментов.

Поскольку проектировщик взаимодействия может оказать серьезное влияние на создание тех или иных фрагментов программы, в то же время избегая деталей реализации, мы считаем проектирование процессом определения продукта. По сути дела, проектировщики определяют внутреннюю часть продукта путем определения внешней. Взаимодействие с пользователем описывается очень подробно и очень точно, тогда как вопросы реализации остаются на совести программиста.

Проектировочные документы приносят пользу программистам

Помните вопрос «Когда проект *готов?»* из главы 3 «Пустая трата денег»? Основное назначение документов, созданных проектировщиком взаимодействия, в том, чтобы ответить на него. В общих чертах проектировочный документ – это надежный механизм контроля процесса программирования. Он – то же самое, что сценарий и раскадровка для производства фильма, он проясняет для всех участников принципы работы, материалы, необходимые для

создания и применения, а также условия, выполнение которых позволяет сказать, что работа завершена. Как правило, в процессе разработки труднее всего контролировать этап написания кода, более других связанный с рисками, поэтому непонимание «готовности» обходится довольно дорого.

Прозрачный документ помогает руководству компании точнее понять, что создает компания, и таким образом сконцентрировать усилия предприятия в нужном направлении. Руководство получает в свое распоряжение более точное определение продукта, подходящее для инвесторов, партнеров, сотрудников и коллег. Это гарантирует, что все рабочие усилия компании направлены на достижение одной цели.

Программистам нужен сильный и умный лидер. В конце концов, они сильные и умные, они предпочитают, очевидно, чтобы их возглавляли равные по рангу, а не нижестоящие. Как утверждает Джерри Вайнберг (Jerry Weinberg), всем известно, что «плохих руководителей найти проще, чем хороших программистов». Это ставит хорошего программиста более выгодное положение, несмотря на номинальный авторитет руководителя в корпоративной иерархии.

Руководитель разработки продукта слаб, если не способен точно и убедительно описать то, что создает его команда. Как правило, руководство выражает свои желания с помощью ничего не значащих критериев фиксированных сроков сдачи и с помощью торговли функциями. О слабости таких инструментов мы уже говорили. Лишь программистам доподлинно известно, как будет выглядеть законченный продукт, поэтому они в большей степени властны над проектом, чем руководитель.

Я работал с программистами, которые испытывали неприязнь к выполнявшей проектирование сторонней компании. Они знали, что моя работа заключается в «проектировании» и что это самая творческая и самая интересная часть *их* работы. Однако поработав с нами, они поняли, что мы не только не отнимаем у них работу, но и делаем ее более качественной.

Недавно мне довелось побывать на одном собрании, где присутствовали язвительные программисты клиента, не предупрежденные о нашей роли. Один седобородый программист по имени Фред выступал особенно энергично. Стоило нам высказать идею или представить какой-либо способ отображения информации для пользователя, Фред нападал на нас. Он очень умен и излагал свои мысли четко и громко. Всякий раз когда мы демонстрировали слайд, иллюстрировавший изменение взаимодействия, он закатывал глаза, снисходительно улыбался и отпускал замечание по поводу того, что мы не осознаем, «каких героических усилий это потребует» от него и от его команды. Он все время давал понять, что наши решения не облегчают, а осложняют его работу.

Под конец, когда собравшиеся стали разбредаться, нашей команде удалось поговорить с Фредом наедине. Мы объяснили ему, что наша задача сделать программу проще и мощнее с точки зрения *конечного пользователя,* и что мы полностью осознаем: наши решения требуют серьезных дополнительных размышлений на уровне программирования. Мы настаивали, что нас интересует только конечный пользователь. Внезапно на лицо Фреда нашло выражение потрясения. Он воскликнул: «Вы бросаете мне *серьезный технологический вызов!»* Его отношение полностью изменилось, как только он осознал, что мы принесли ему грааль всех программистов – сложную проблему, достойную решения.

Никоим образом не пытаясь ему угрожать, мы дали ему то, чего он больше всего желал: шанс еще раз проявить себя, как самого умного, самого изощренного, самого опытного программиста. Его отношение изменилось, как только он осознал, что мы отняли лишь запутанный человеческий аспект программирования, который ему так не нравился, а ему осталось лишь одолеть чисто алгоритмическую начинку программы. Мы стали для него благодетелями, а не врагами.

Проектировочные документы идут на пользу маркетингу

В большинстве некомпьютерных предприятий на этапе определения продукта работают профессиональные маркетологи. В бизнесе программного обеспечения маркетологов исключили из процесса. Здесь им достаются лишь запросы новых возможностей. Если они требуют исправления дефектов, программисты попросту швыряют им в лицо жесткий график проекта с вопросом: «И как я могу что-то исправить, если вы не даете мне времени?» Руководитель отдела маркетинга не смеет потратить это драгоценное время, потому что не просто нарушит этим график, но и покажет всем, насколько график лжив, и программисты в будущем смогут безнаказанно заваливать все сроки. Маркетологи знают, что список функций – слабый механизм, и часто стремятся к участию в процессе определения продукта. К сожалению, маркетологи, похоже, неспособны давать указания, которые показались бы программистам полезными или осмысленными.

Одно из наиболее важных преимуществ усиленного процесса проектирования, а также доскональной документации, как части этого процесса – польза, которую извлекают маркетологи. Маркетологи рассказывают проектировщикам, какие потребности или желания пользователей надеются удовлетворить. Проектировщики взаимодействия изучают таких людей, чтобы определить их цели и создать набор персонажей. Точные определения персонажей пользователей – важная часть проектировочной документации, и эти определения становятся точкой фокусировки усилий по маркетингу. Программисты работают только с кодом, но персонажи одушевляют этот код. Маркетологи работают с каналами, рынками, средствами массовой информации, реселлерами, но персонажи одушевляют все эти сущности. Наконец, у программиста и маркетолога появляется общая точка отсчета.

По существу, проектировщики взаимодействия играют роль посредников для профессиональных маркетологов. Проектировщик – это человек, способный переводить с языка маркетологов на язык программистов. Когда у маркетологов нет четкого понимания, они могут описать свои заботы проектировщикам, а те разовьют мысль в терминах персонажа. Затем проектировщик сможет перевести проблему в спецификацию взаимодействия. Более того, маркетолог увидит, как выполняются его запросы, и может быть уверен, что ему не придется искать покупателя для голого инженерного продукта.

Разработка персонажей пользователей – дело, как правило, очень знакомое профессиональным маркетологам. Они часто выполняют подобные упражнения, определяя персонажи покупателей продукта. При этом изучаются каналы распространения и демография вместо целей и сценариев, поэтому персонажи обычно получаются иными, однако всегда оказывается, что они полезны маркетологам для планирования. Маркетологи получают возможность четко изложить *покупателю,* каким образом продукт облегчит жизнь *пользователям.*

Проектировочные документы помогают в разработке документации и технической поддержке

Любой технический писатель скажет вам, что качественное проектирование избавляет от необходимости писать невероятные объемы документации. Чем меньше сложного взаимодействия, тем меньше пространных объяснений. Авторы документации получают возможность больше времени потратить на более высокие уровни взаимодействия. Вместо того чтобы, за руку водить пользователя по запутанным джунглям непонятного интерфейса, они могут выйти на другой уровень и потратить силы на более интересные пользователю вопросы решения проблем предметной области. Вместо рассказа о том, где хранятся файлы системы инвентаризации, документация может поведать о принципах инвентаризации, что определенно принесет больше пользы.

То же верно и для технической поддержки. Чем лучше проектирование, тем меньше звонков от пользователей. Как показало проектирование сканера Peacock, описанное в предыдущей главе, проектировочный документ радикально снижает потребность в

технической поддержке.

Проектировочные документы помогают руководителям

Из всех участников наибольшую выгоду от проектировочных документов получают руководители разработкой продуктов. Упреждающее описание того, что необходимо создать, ускоряет весь процесс разработки, делает его более взвешенным, менее рискованным, менее дорогостоящим. Эффективность процесса разработки повышается, и он больше не управляется клиентами.

Прежде всего, проектирование означает большую предсказуемость. Проектирование продукта означает, что фаза программирования будет более предсказуемой. Кроме того, и успех продукта становится легче прогнозировать и оценивать. Это два наиболее рискованных и дорогостоящих аспекта разработки программных продуктов. Они сокращают стоимость производства и побеждают миф о непредсказуемости рынка. Эд Форман, вице-президент по разработке продукта Elemental Drumbeat, говорит:

> «Выгоды от услуг проектирования измеряются сэкономленными месяцами напряженной работы».

Проектировочные документы выгодны компании в целом

Какова сущность построения успешного бизнеса? Все участники должны работать вместе для достижения одной цели. Любая неразбериха или разлад в целях рассеивает усилия двояко. Во-первых, вы теряете усилия тех, кто идет в неверном направлении, а во-вторых, их усилия сдерживают тех, кто пытается идти в нужном направлении. Если один человек в каноэ гребет в противоположном направлении, команда уже не может всерьез претендовать на призовое место. Успех требует гребли в одном направлении, и любая сила, отвлекающая внимание одного человека, наносит вред всем.

Более того, точно осознавая, что вы делаете, вы избежите траты сил на то, чего вы точно *не* собираетесь делать. Ни одна компания не может себе позволить тратить силы на вещи, не относящиеся к делу.

Избавляясь от управления, ориентированного на фиксированные сроки сдачи, и торга по поводу функций продукта, документальное описание продукта направляет все внимание компании на его качество, которое неизбежно радикально повышается. Качество, в свою очередь, умножает запасы самого бесценного актива компании: лояльности клиентов.

Кто отвечает за качество продукта?

Если за качество продукта отвечает каждый, это означает, что за качество продукта не отвечает никто. Слишком уж легко предположить, что решением проблемы качества займется ваш коллега, пока вы работаете над чем-то еще. Программисты несут ответственность за уничтожение всех дефектов кода. Отдел продаж отвечает за закрытие сделок, а маркетологи – за упаковку и позиционирование. Однако в настоящий момент нет лиц, отвечающих за качество и пригодность продукта. Иногда этим лицам не хватает инструментов, чтобы обнаружить и разрешить проблему.

Иногда не хватает умения, чтобы выразить решение. Иногда у них нет авторитета, достаточного для того, чтобы их решение было реализовано. Как мы видели в предыдущих главах, написание кода подрывает способность программиста думать о целях пользователя. Руководителям разработки и без того есть чем заняться, они не могут концентрироваться на деталях поведения продукта. Маркетологам не хватает технической подготовки, и это снижает их способность общаться на технические темы, а значит, подрывает доверие к ним

со стороны программистов. При отсутствии тщательно документированного проекта нет надежды на его правильную и эффективную реализацию.

Главная рекомендация этой книги: *за качество продукта в конечном итоге должен отвечать проектировщик взаимодействия.* Необходимо разрешить ему определять содержание и поведение программы. Он должен работать со списком функций, и даже график разработки, в основном, должен быть на его совести. Он защищает пользователя и должен обладать властью над всеми внешними аспектами продукта.

За всю эту власть проектировщик взаимодействий получает и ряд очень серьезных обязанностей. Без такого сочетания авторитета и ответственности программисты не будут уважать проектировщиков и вернут себе контроль над продуктом. Проектировщики должны быть кровно заинтересованы в успехе. Мандат доверия команды проектирования взаимодействия включает проектирование реальных, простых в применении, привлекательных продуктов, позволяющих пользователям достигать практических целей, не отказываясь от целей личных. Более того, проектировщик взаимодействия должен создавать исчерпывающие письменные спецификации, исходя из которых программисты и будут строить продукт. Проектировщик взаимодействия должен предоставить маркетологам прозрачные описания пользователей и того, как продукт удовлетворяет потребности этих пользователей. И самое важное – он принимает на себя ответственность за качество конечного продукта.

Включение проектирования в процесс

В предыдущей главе мы видели, что многие профессионалы, предлагавшие свою помощь в проектировании взаимодействий, не добились успеха. Речь шла о юзабилити-специалистах, промышленных дизайнерах и других, кто пытался и не смог решить эту проблему. В настоящее время в индустрии не существует группы, способной решить эту проблему.

Ряды проектировщиков взаимодействий медленно пополняются, и здесь следует иметь в виду, что важнее правильно подготовить процесс разработки к включению проектирования, а не нанять самых талантливых проектировщиков взаимодействий. Самое важное – принять решение, что программированию будет предшествовать проектирование. Самый блестящий проектировщик мира ничем не поможет, если продукт на следующей неделе переходит на стадию бета-тестирования.

Многие империи, производящие программное обеспечение, выросли на усилиях очень молодых и очень неопытных программистов. Вероятно, они получали свободу действий в вопросах программирования, а сочетание невероятной ответственности с безграничными полномочиями часто становится плавильным тиглем, рождающим великие вещи. Те же правила действуют в проектировании взаимодействия. Если человек получает ответственность за качество продукта и *соразмерные полномочия,* то он может принять на себя эту ношу независимо от своего опыта. Если взять подходящего человека и дать ему полную власть над качеством и поведением продукта, вы получите продукт намного, намного лучший, чем если этого не сделать. Проблема не в людях, но в процессе. Разумеется, при прочих равных всегда лучше нанять специалиста с подходящим опытом. Однако если специалисты не вписываются в бюджет или просто не существуют, менее опытные проектировщики все равно лучше, чем совершенно неподконтрольные программисты.

Что значит быть «подходящим человеком»? Лучше всего подойдет человек, отдаленный от процесса реализации продукта настолько, чтобы он мог легко ставить себя на место пользователя. Это может быть и программист, но определенно не из числа тех, что будут создавать эту программу. Иначе возникает слишком большой конфликт интересов.

Откуда берутся проектировщики взаимодействия

Как бы там ни было, вам необходимо выбрать проектировщика взаимодействий. Начав искать, вы обнаружите разочарованных проектировщиков взаимодействия практически в каждой высокотехнологической компании: это технические писатели, к которым программисты обращаются за помощью; менеджеры продуктов с полками, заставленными книгами по дизайну интерфейсов; эргономисты, заговаривающие о более ранней интеграции в разработку; маркетологи, подчеркивающие, что прикупили стереосистему с минимально возможным количеством кнопок; программисты, которые пишут очень мало кода, но с которыми стремятся работать другие программисты. В действительности, когда в компании станет известно, что какой-то проект начнется с фазы проектирования взаимодействия, наверняка кто-то *вызовется* отвечать за качество продукта.

При найме проектировщика на полный рабочий день хороший кандидат не обязательно назовет себя проектировщиком взаимодействия. Вам нужны люди, понимающие, какие ограничения накладывает техническая сторона дела, и горящие желанием проектировать, и таких людей, обладающих самой разнообразной подготовкой, можно найти в самых различных средах. Нанимая людей в свою группу проектирования, я прошу их пройти испытание на конкретной задаче, поскольку знаю, что сведения в резюме бывают весьма разнообразные. В моей студии несколько проектировщиков в прошлом были техническими писателями, руководителями разработки, сотрудниками технической поддержки, дизайнерами. Многие из моих проектировщиков имеют степени в гуманитарных областях, но есть также люди со степенями в физике, архитектуре, информатике и промышленном дизайне.

Опыт работы в технической поддержке или документировании дает проектировщикам перспективу для оценки потребностей типичного пользователя. Менеджеры продуктов знают о нуждах и заботах программистов в процессе разработки. Дизайнеры и промышленные дизайнеры страстно увлечены элегантным дизайном и обладают способностью такой дизайн создавать. Проектировщики с гуманитарным образованием, работавшие в области высоких технологий, сочетают познания в технологии со способностью ясно выражать свои мысли.

Создание команд проектировщиков

Обсуждение способов организации и управления командой проектировщиков может занять целую книгу. Здесь же я затронул лишь некоторые методы проектирования, чтобы прояснить, что имею в виду под «проектированием», но не пытался изложить методологию проектирования целиком. Тем не менее, мой опыт управления командами проектировщиков дал возможность выделить несколько ключевых принципов.

Команды должны быть небольшими. Чтобы продвигаться вперед, проектировщикам необходимо общее видение. Для каждого продукта я отряжаю команду из двух или трех человек, которым время от времени могут помогать и содействовать другие специалисты. В сложных проектах могут возникать ситуации, когда у продукта появляется несколько различных интерфейсов, и в этот момент можно разделить проблему на несколько частей: каждой отдельной команде по части. Но до этого момента у семи нянек дитя будет без глаза.

Изолируйте команду от руководителей и программистов. На старте проекта проектировщикам будет необходимо поговорить с другими людьми, работающими над продуктом, чтобы получить четкую формулировку проблемы и дать определения персонажам. После этого проектировщикам требуется независимость в исследовании тупиковых путей; так они получают наилучшие решения, и делать это могут только в уединении.

Назначьте человека, отвечающего за документирование работы команды. Все участники вносят вклад в создание документации, но у нее должен быть «владелец», придающий эффективность процессу.

Дайте команде время, чтобы собраться с мыслями. В разгаре проекта, когда основные

проблемы уже решены, имеет смысл ответить на конкретные вопросы команды. В начале проекта команда должна тщательно все продумать и представить свои соображения в виде стройной концепции. Когда моя студия полностью прорабатывает структуру продукта для клиента, мы предоставляем документацию и свои проекты с неформальными контрольными точками по мере необходимости. На первой презентации мы очерчиваем проблему, представляем персонажи и описываем, какие задачи должно решить проектирование. На каждой из последующих презентаций мы более подробно описываем проект продукта.

Управляемый процесс, сосредоточенный на проектировании вместо программирования, позволяет компаниям избежать игры с огнем высоких технологий. Они заранее могут узнать, что должно понравиться пользователям и как это обеспечить. Они будут знать, когда процесс разработки завершен, а у специалистов различных дисциплин появится единое, объединяющее видение продукта.

Глава 14
Мощь и удовольствие

Чтобы ваш бизнес получил все выгоды от проектирования взаимодействия, эту дисциплину необходимо сделать составной частью процесса разработки. Ее нельзя добавить задним числом.

В предыдущей главе я писал, что проектирование необходимо излагать на бумаге, прежде чем начнется создание кода. Однако в кипящем котле разработки продукта программист по-прежнему имеет возможность игнорировать проектировочный документ, независимо от качества документа. Такое развитие событий весьма вероятно в пассивно-агрессивной культуре разработки программного обеспечения, где инженеры считают любые проектировочные вводные не более чем советом, которому можно следовать, если позволяет рабочая нагрузка и есть желание.

Следует четко и ясно дать понять всем участника проекта, что проект – это чертеж, которому необходимо следовать, а не просто предложение. Если приверженность проектированию не демонстрируется энергично и открыто, разработчики будут предполагать, что лишь на них возложена ответственность за создание успешного продукта.

Есть только один способ эффективно передать эту мысль. Высшее руководство компании должно недвусмысленно заявить всем менеджерам по проектированию и разработке, что программисты освобождаются от ответственности. Оно должно дать понять, что за качество продукта отвечает теперь команда проектировщиков, и проектировщики наделяются полномочиями требовать, разумеется, под присмотром руководителей.

Программисты вольны импровизировать внутри программы, но любой аспект взаимодействия с пользователем, имеющий четкое определение, должен быть реализован строго по описанию. Описание можно подвергать сомнениям, но нельзя в одностороннем порядке игнорировать или переиначивать. Предписания проектировщиков не следует считать советом, который можно воспринимать частично или видоизменять.

На команду проектировщиков возлагается ответственность за все, с чем вступает в контакт пользователь. Не только за программное, но и за аппаратное обеспечение. Следует принимать во внимание и сопутствующие программные модули, такие как программы установки.

Это, вероятно, наиболее радикальное требование, выполнение которого необходимо для успешного проектирования, причем такое, которое требует наибольшей культурной адаптации. Позже в этой главе мы обсудим культурные вопросы более подробно. А сейчас рассмотрим пример компании, успешно включившей проектирование в процесс разработки.

Пример налаженного проекта

Моя студия проектирования недавно завершила работу над одним из самых успешных проектов. Клиент – небольшая компания Sun Healthcare Systems, Inc., создающая программное обеспечение для управления всевозможными аспектами работы учреждений системы здравоохранения.

На первых встречах я старательно объяснял заказчику важность персонажей и рассказывал о том, какую роль мы им отводим в процессе проектирования. К нашему большому удовольствию и удивлению, команда SHS восприняла эту концепцию очень благосклонно. На первое проектное совещание они принесли собственный набор из десятка уже определенных персонажей. Нам все же пришлось потратить время на изучение предметной области, чтобы проверить качество персонажей и доработать их, зато полностью исчезла необходимость убеждать разработчиков и маркетологов в применимости персонажей, как инструмента.

Бизнес SHS приводит эту компанию к тому, что Мишель Борк (Michel Bourque) из компании Clinidata, расположенной в Монреале, называет «Клиническим водоворотом». Кабинеты врачей попали в число первых объектов компьютеризации в малом бизнесе, однако преобразованию поддалась лишь финансовая часть. Та же сторона, где врач взаимодействует с пациентом, упрямо сопротивлялась пришествию цифровой эры, и остается одним из последних бастионов некомпьютеризованного мира.

Усилия SHS в основном направлены на администрирование, но существенная часть задач оказывается прямо в этом водовороте. Мы участвовали в небольших проектах других компаний, работающих в этой области, но в самом центре водоворота мы еще не были. Возможность работать над столь серьезным и сложным проектом нас сильно воодушевила.

Воодушевились и в компании SHS, но для начала сообщили нам, что масштабы бизнеса настолько велики, что компания сомневается в нашей способности эти масштабы осилить. SHS считала, что ее бизнес просто слишком велик и сложен для понимания. Для нас это был вызов, и мы приняли его с готовностью.

Проект был *большой.* Мы выделили *пять* ключевых персонажей. Ни в одном из прошлых проектов не бывало больше трех. Поначалу мы с подозрением отнеслись к такому числу, однако, пересмотрев результаты, поняли, что SHS действительно пытается охватить огромный сегмент рынка здравоохранения. Разумеется, создание программного обеспечения сразу для пяти ключевых персонажей – слишком крупный проект. В SHS это поняли, поэтому продукт проектировался и создавался поэтапно, по одному персонажу на этап.

Дэвид Вест (David West), вице-президент по разработке и наш связной в SHS, помимо прочего снискал доверие и уважение других сотрудников этой растущей организации. Маркетологи знают, что он действует в их интересах. Знают это и программисты. Знают, что он суров, но справедлив. Он подобен камню в бурлящих водах разработки. Он верен процессу проектирования, и другие разработчики стали доверять нашей работе, воспринимать ее всерьез, как спецификацию.

Когда SHS вступила в контакт с Cooper Interaction Design, ее отдел разработки программ был организован в соответствии с уже имевшимся программным продуктом. А продукт имел два модуля – клинический и финансовый.

Проведя исследование и разработав персонажи, мы быстро поняли, почему существующая система не способна удовлетворить медиков. Если даже не учитывать серьезные проблемы взаимодействия, деление между информационными подсистемами (клинической и финансовой) было надуманным. Это порождало лишнюю бумажную работу, которую пользователь был вынужден выполнять, чтобы обойти недостатки системы обработки данных. Каждый пользователь обитал на собственном островке данных, поскольку два модуля системы не были взаимосвязаны.

Мы рекомендовали создать объединенную медицинскую карту пациента, содержащую как клиническую, так и финансовую информацию для консолидированной базы данных, а также модульный пользовательский интерфейс, позволяющий каждому персонажу получать доступ к информации, необходимой для решения ее задач. В результате SHS

перепроектировала базу данных, лежащую в основе продукта. И что примечательно, реорганизовала сотрудников отдела разработки в соответствии с этой новой архитектурой! Разработчики сформировали две новые группы. Одна из них работала с архитектурой медицинской карты и базой данных, а вторая – с интерфейсами персонажей. Все дальнейшие программные спецификации и документы в SHS будут содержать имена персонажей для четкого определения функций.

Программисты SHS поступили мудро, откладывая программирование до завершения работ по проектированию. Дэвид и команда SHS хорошо понимали, что простой программистов обходится недешево, но гораздо дороже обходится заливка бетонной смеси программного кода не там, где требуется.

Программисты работали над логикой базы данных, не затрагивающей взаимодействие с пользователем. Кроме того, они разбили проект на несколько фаз, среди которых был и перевод существующей версии продукта на более высокий уровень. Таким образом, программистам было чем заняться, и конфликтов с долгосрочными стратегическими планами не возникало.

Для того, чтобы гладко синхронизировать нашу работу с работой программистов, мы разделили процесс на несколько крупных частей.

Мы договорились, что на начальном этапе проектирования будем уделять внимание лишь двум персонажам из пяти ключевых, а к трем остальным вернемся позже. Опять же, это позволило вести проектирование в опережающем темпе, и в то же время разработчики не сидели без дела.

Осознанное проектирование взаимодействия

В большинстве компаний проектирование основного продукта или услуги считается важнейшим умением. В мире высокотехнологических продуктов, основанных на программном обеспечении, считается (и ошибочно), что проектирование продукта находится в компетенции инженерного персонала. В действительности акт творения состоит из двух частей: проектирования и программирования. Готовность разрешить проектировщикам взаимодействия работать с важнейшим элементом бизнеса наряду с инженерами требует значительных культурных перемен.

В любой компании, независимо от ее специализации, сотрудники знают, что имеют определенные обязанности. Так, в компании, производящей катушки для динамиков, руководитель производства знает, что хотя его задача заключается в покупке лучшего сырья по наинизшей цене, он не сможет подписать контракт с поставщиком, пока не получит одобрение совета компании. Руководитель производства не очень разбирается в юридических вопросах контрактного дела, однако знает, что не стоит создавать трудности своей компании, пренебрегая советом профессионалов в области контрактов и сделок. Даже не разбираясь в контрактном деле, точнее, как раз поэтому, руководитель знает, что необходимо привлечь юриста.

Приемщик в грузовом доке, обладая самым низким рангом, знает, что уполномочен расписываться лишь за грузы, доставка которых уже оговорена, но не уполномочен расписываться за иные.

Основатель и президент компании, выпускающей катушки, также вполне осознает необходимость участия юристов на всех уровнях. Он не имеет профильного юридического образования, а потому советуется с помощником, прежде чем подписывать какие-либо документы.

Ни один из этих людей не обладает специальными знаниями в области юриспруденции, но каждый полностью осознает важность участия юриста. Никто в компании ничего не подпишет, пока юристы не подтвердят, что это можно сделать. Сама по себе компания осознает необходимость в таком присмотре и даже, временами, вмешательстве.

Такое осознание, распространяющееся на всю компанию, верно и для других областей.

Когда катушечной компании потребовалось новое производственное здание, она наняла стороннего профессионала, архитектора. Руководитель производства и президент хорошо разбираются в особенностях производственных мощностей, однако знают, что их понимание нюансов конвейерной работы и строительства зданий поверхностно. Никому в этой компании и в голову не придет расширять производство, не посоветовавшись сначала с архитектором. Архитектор переводит потребности пользователей в термины, понятные строителям.

То же верно и для рекламы. Руководитель отдела маркетинга не будет просить рабочего описать выгоды продукта для брошюры или для профильного отраслевого журнала. Каждый в компании, независимо от опыта, понимает, что реклама – дело профессионалов, и что специалисты по рекламе могут обеспечить компании отличный пиар. Разумеется, эти специалисты могут быть сотрудниками компании, как могут быть и наняты в рекламном агентстве. Оба варианта хороши.

Приведенная аналогия не идеальна. Ни архитектура, ни юридическая консультация для производственной компании не лежат в сфере основного занятия. Однако программирование – это создание продукта, а именно создание продукта, как правило, и считается занятием компании. Учитывая непосредственное влияние продукта на бизнес, можно ожидать, что любая компания будет вдвойне осторожна, чтобы бразды правления не попали в плохие руки, – еще более осторожна, чем в случае с рекламой, архитектурой или приобретением чего-либо.

Мы должны сделать так, чтобы в компании поняли: проектирование взаимодействия представляет собой область, требующую профессиональных навыков, и что интерактивные продукты нельзя просто конструировать инженерными способами, их следует еще и проектировать, чтобы добиться успеха на открытом рынке.

Польза от перемен

В мире программного обеспечения так много апологетов, а их влияние настолько велико, что власть этой касты ослабевает очень медленно. Но эта власть ослабнет обязательно. Для этого достаточно, чтобы люди поняли: технологии не обязательно должны быть бесчеловечными. Чем чаще пользователи будет сталкиваться с программами, не унижающими их человеческое достоинство, тем быстрее они будут терять терпение, встречаясь с иными программами, которые оскорбляют и раздражают их. Они пинками погонят танцующих медведей прочь.

Когда пользователей таких продуктов было немного, все они были и непосредственными участниками процесса, а потому понимали, насколько революционна эта технология. По мере проникновения технологии во все сферы жизни всё меньше причастившихся осознают, насколько это достижение велико. Они не готовы простить продукту некачественное взаимодействие только потому, что создание этого продукта отняло много сил.

Следование за технологией кажется идеей неплохой, однако результатом обычно становятся утомляющие продукты, более сложные наследники тех, что уже были созданы. Проектирование взаимодействия позволяет вырваться из этого круга и создавать продукты, которые делают то, что никогда раньше не делалось.

Проектирование взаимодействия делает продукт привлекательным, награждая уникальным рыночным преимуществом – преданностью покупателей. Если покупателя осчастливил ваш продукт, этот человек надолго останется клиентом вашей компании и вашего брэнда. Если же ваш продукт – очередной танцующий медведь, клиенты будут оглядываться в поисках более простых и дружелюбных альтернатив.

Проектирование взаимодействия может сократить затраты времени на разработку продукта. Если вы заранее знаете, что именно следует создавать, то потратите меньше времени на поиски верного пути.

Создание правильного продукта – всегда итеративный процесс. Чтобы добиться точности в деталях, требуется несколько попыток. Проектирование взаимодействия позволяет значительно уменьшить количество итераций. Выпуск каждой новой версии продукта сопряжен с огромными затратами, поэтому, уменьшая количество версий с четырех до двух, вы экономите массу денег и времени.

Процесс разработки удешевляется, если версий становится меньше и код выбрасывается не в таких количествах. Программисты часто жалуются, что наши проекты требуют создания более сложного кода, и зачастую они правы. Однако в целом размер кода получается обычно меньшим. Стоимость кода не сильно растет с увеличением сложности, однако значительно растет с увеличением *объема.* Каждую лишнюю строку кода необходимо тестировать, отлаживать и поддерживать.

Почему они не едят пирожных?

Я живу и работаю в Кремниевой Долине, штат Калифорния. Практически все знакомые мне люди вовлечены в индустрию высоких технологий. Мы все обеспечены, образованны, географически и социально мобильны, замечательно управляемся с компьютерами, сотовыми телефонами, видеомагнитофонами, банкоматами и всеми прочими представителями зверинца продуктов, основанных на программном обеспечении. Когда я обедаю в Crescent Park Grill или Spago, посетители за соседними столиками все время обсуждают клиент-серверные архитектуры и веб-интерфейсы. Восхитительное место для жизни, но оно не дает представления о большинстве жителей этой страны, не говоря уже обо всей планете. Здесь, в Долине, наши оценки качества высокотехнологических продуктов могут искажаться легко и радикально. Мы забываем, насколько сложно в действительности применять эти продукты.

Десять лет назад консультант по розничным продажам Сеймур Меррин (Seymour Merrin) сказал, что нам проще было убедить покупателей, что программами легко пользоваться, чем сделать так, чтобы программами было легко пользоваться. Высказывание Меррина было циничным, однако он также выражал удивление, что нам сошла с рук столь наглая ложь. Сегодня его слова все еще справедливы, но рост высоких технологий не оставляет возможности и дальше выезжать на одном цинизме – нам требуется настоящее решение.

Люди знают, что компьютерами очень трудно пользоваться, однако предполагают, что тому есть причины. Большинство считает, что лучше уже и быть не может.

Простые пользователи, не вовлеченные в индустрию компьютеров, приходят в крайнее раздражение из-за сложных в применении продуктов, тогда как люди, занятые созданием продуктов, в массе своей удовлетворены состоянием дел. Программистам не кажется, что компьютерами сложно пользоваться, поэтому они готовы терпеть некоторые вещи, пока есть возможность поиграть в технологические игры, создавая новых классных танцующих медведей.

Что до остальных из нас, мы получаем программы, соответствующие требованиям, а требовали мы до сих пор очень немного. Разработчики программ засыпают нас штучками, примочками, возможностями, которые нам не нужны, которые мы никогда не применяем, но вынуждены оплачивать. Мы требуем, чтобы программы не сбоили, поэтому программы подвергаются исчерпывающему тестированию и, как следствие, обладают разумной надежностью. Мы требуем новых версий немедленно, и компании выпускают версии с безумной скоростью. Но, не задумываясь о том, что жизнь может стать лучше, мы не требуем, чтобы продукты стали мощными и приятными, поэтому продукты и остаются слабыми, угнетающими.

Время от времени у потребителей появляются призрачные надежды, что следующая волна технологии, скажем, распознавание голоса, сделает программные продукты простыми в применении. Надеяться на это наивно и неумно. Мне грустно видеть, как апологеты грубо

подыгрывают таким надеждам.

Программное обеспечение уместно сравнить с пластилином – оно может принимать любую форму, какая будет угодна авторам. Они не создают простые в применении программы не потому, что это невозможно, а потому, что не умеют. Чтобы не признавать этот неудобный факт, они заявляют, что лучше сделать просто невозможно «по техническим причинам». Пользователи компьютеров, программистами не являющиеся, вынуждены соглашаться со специалистами и страдать, или же не соглашаться со специалистами и – вот именно – все равно страдать. Не будучи специалистами, они не способны предлагать собственные решения, поэтому к ним относятся, как к бесполезным нытикам.

Штат Детройт производил гигантские хромированные прожорливые автомобили и лицемерно утверждал: «Мы даем потребителям то, что им нужно». Во время нефтяного кризиса семидесятых японцы вышли на рынок с экономичными небольшими автомобилями и нанесли Детройту удар, который не забудется никогда. Сегодня автомобильная индустрия Америки проявляет гораздо большее уважение к желаниям потребителя и уже не осмеливается утверждать, что все знает лучше.

Япония захватила позиции на автомобильном рынке, выполнив желания пользователей, о которых те и не подозревали. При этом потребители способны отличить хорошую вещь от плохой, если им дадут ее увидеть. Точно так же высоты в проектировании программного взаимодействия сегодня остаются свободными, они не захвачены никем. Microsoft уязвима не меньше, чем General Motors в 1974 году.

Массовый рынок потребителей, не знакомых с технологическими тонкостями, охотно принимает простые в применении продукты, что и подтверждает взрывной рост среды Web. Люди, привлеченные простотой ее использования, точно так же оценят качественно спроектированные продукты, делающие сложные вещи простыми для пользователей.

Потребители, не знакомые с тонкостями технологии и не принадлежащие к анклавам вроде Кремниевой Долины, не будут требовать изменений просто потому, что не могут образовать сплоченную группу. Разумеется, хорошее от плохого они отличают, но только после того, как хорошее и плохое появляется на полках магазинов.

Изменения произойдут только тогда, когда люди, принимающие участие в процессе и способные повлиять на ранние стадии развития продуктов, заинтересуются решением этой проблемы. Программисты вовлечены в конфликт интересов, а потому мой призыв адресован апологетам в самом сердце этой индустрии высоких технологий. Современные бизнесмены вовлечены в жизнь этой индустрии, хотят они того или нет. Вряд ли еще остались в мире предприятия, не вставшие на путь внедрения информационных технологий.

Ни один из современных нам продуктов, основанных на программном обеспечении, не способен дать мощь и удовольствие от применения людям, не входящим в число одержимых технологией. Инженерное же сообщество просто сообщает, что пользователям придется получить «компьютерное образование». Полагаю, в историю эта фраза войдет наравне со знаменитым и снисходительным: «Почему они не едят пирожных?» Марии-Антуанетты. Французская революция дала народу хлеб, а грядущая революция проектирования даст народу технологию.

Изменить процесс

Большинство разработчиков программного обеспечения и технических руководителей делают то, что делают, потому что верят в процесс, однако при этом не считают процесс догмой. В них достаточно прагматизма, чтобы принять изменения, когда они увидят, насколько эффективно проектирование взаимодействия. По моим наблюдениям, наглядная демонстрация выгод проектирования вызывает у этих людей желание интегрировать проектирование в процесс разработки.

Разработчики программного обеспечения известны способностью менять свои убеждения. Конечно, они инженеры, их мышление навсегда останется инженерным, однако

они воспринимают новые (и даже контрастирующие) подходы, если доказана эффективность этих подходов.

Двадцать лет назад в бизнесе программирования считалось нормальным, что разработчики самостоятельно тестировали свой код. Более того, программист мог естественным образом предполагать, что только он сам способен достаточно надежно протестировать собственный код, только он сам может знать обо всех слабых местах и пыльных углах, куда следует заглянуть.

Удивительно, но факт: в те времена все знали, что программисты, хотя им и приходилось заниматься тестированием, как один ненавидели это занятие и сожалели о потраченном на него времени и силах. Однако программисты выполняли работу по тестированию, поскольку честно верили в свою роль в процессе, как и в необходимость агрессивного тестирования.

В последние два десятилетия все в этой индустрии постепенно осознали, что эту часть работы могут выполнять подразделения профессиональных тестеров, освобождая от нее программистов. Поначалу программисты отнеслись к идее скептически, но затем оценили ее. Тестерам же, к нескончаемому удивлению большинства программистов, действительно *нравилось* тестирование. Они получали удовольствие, создавая новые и еще более изощренные инструменты для испытания продуктов, находя слабые места и пробелы, тестируя специальные и вероятные случаи. Конечно же, скрупулезное изучение продукта профессиональными тестерами намного лучше. Программисты обнаружили, что не просто избавились от большой и неприятной составляющей своей работы; теперь эта работа выполнялась более надежно, своевременно, более продуманно и организованно. Современная доктрина разработки программного обеспечения гласит, что правильное отношение числа тестировщиков к числу программистов – один к одному. Сегодня уже не найти программиста, который настаивал бы, что является лучшим тестером своего кода.

Мы увидим подобный постепенный переход, когда проектирование взаимодействия станет частью процесса разработки. Наибольшие дивиденды получат те последователи проектирования, которые раньше других начнут применять этот подход.

Цели у разработчиков программного обеспечения и проектировщиков взаимодействия общие: и те и другие хотят, чтобы продукт стал успешным. Просто их инструменты и термины для измерения успеха коренным образом различаются.

Если у программиста нет убедительных доказательств, он всегда отступает под защиту своей подготовки, опыта, интуиции. Интуиция подсказывает, что функций должно быть как можно больше. Опыт подсказывает, что нельзя позволять дилетантам вмешиваться в чувствительный, сложный, тонкий процесс разработки со всякими причудами и догадками. Подготовка подсказывает, что интерфейсы следует конструировать, основываясь на своих представлениях.

Проектировщик взаимодействия не может прямо критиковать эти мотивы. Разработчики мыслят слишком рационально, чтобы променять свой опыт на чужое мнение. Проектировщик должен показать им новый взгляд на проблему, должен показать им, что этот взгляд эффективен и совместим с уже существующими идеями.

Независимо от твердости позиции проектировщика взаимодействий, весьма маловероятно, что он обладает лучшим пониманием внутреннего устройства программы, чем программист. Иными словами, он не может смотреть на проблему глазами программиста. Чтобы добиться успеха, он должен иметь другую точку зрения.

Точность и полнота проектировочной спецификации – вот точка соприкосновения проектировщиков и программистов. Когда проектировщики дают абсолютно верные решения, программисты начинают доверять и полагаться на них.

* * *

В одном из номеров еженедельника «Business Week»[43] за 1998 год ведущий колонки Стивен Уайлдстром поднял тему раздраженных пользователей компьютеров. Ответы читателей поразили его полярностью мнений, их количеством, накалом страстей. Он сделал вывод:

> У компьютерной индустрии множество озадаченных, раздраженных, несчастных клиентов. Это гораздо более серьезная угроза благосостоянию сектора высоких технологий, чем азиатский кризис, ошибка двухтысячного года и большинство других угроз.

Он процитировал весьма знакомые нам болезненные жалобы уцелевших: «Моя машина заставляет меня думать, что я идиот!» и такие же знакомые вопли апологетов: «Пользователи не знают, чего хотят, а когда знают, то каждый хочет своего. Пусть лучше читают руководства и изучают программы». Стивен завершил колонку так:

> В этих излияниях чувств кое-кого не хватает. Мне написали инженеры, программисты, гуру юзабилити. Однако проектировщики продуктов и маркетологи, принимающие ключевые решения по аппаратной и программной части, подозрительно промолчали. Да у вас тут полно злых клиентов. Как вы собираетесь отвечать?

И действительно, как вы собираетесь отвечать?

[43] Stephen H. Wildstrom «They're Mad as Hell Out There», Business Week, October 19, 1998.