ПРОФЕССИОНАЛЬНО

Стив Макконнелл

ПРОФЕССИОНАЛЬНАЯ РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

СИМВОЛ

Professional Software Development

Shorter Schedules
Higher Quality Products
More Successful Projects
Enhanced Careers

Steve McConnell

Addison-Wesley

ПРОФЕССИОНАЛЬНАЯ РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Сокращение сроков
Повышение качества продукта
Больше удачных проектов
Расширение возможностей успешной карьеры

Стив Макконнелл

Санкт-Петербург — Москва
2007

Серия «Профессионально»

Стив Макконнелл

Профессиональная разработка программного обеспечения

Перевод В. Агапова

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научные редакторы	*А. Сапегин, О. Цилюрик*
Редактор	*В. Овчинников*
Художник	*В. Гренда*
Корректор	*О. Макарова*
Верстка	*Д. Орлова*

Макконнелл С.
Профессиональная разработка программного обеспечения. – Пер. с англ. – СПб.: Символ-Плюс, 2006. – 240 с., ил.
ISBN 5-93286-085-5

Стив Макконнелл, автор бестселлера «Совершенный код», других книг и многочисленных статей о разработке ПО, убедительно показывает, что разработка ПО может быть стабильно успешной, если сделать совершеннее саму профессию разработчика ПО. Он не только показывает, почему и как отрасль пришла к своему современному состоянию, и описывает шаги, которые должен предпринять каждый, кто хочет подняться на новый уровень в создании ПО. Он также говорит о корпоративных методиках, призванных увеличить количество профессионально выполненных проектов, и о лицензировании организаций и академических учебных программ как о средстве повышения профессионализма и отдельных разработчиков, и в индустрии ПО в целом.

ISBN 5-93286-085-5
ISBN 0-321-19367-9 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 31.08.2006. Формат 70x90$^{1}/_{16}$. Печать офсетная.
Объем 15 печ. л. Тираж 3000 экз. Заказ N
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

На высокую башню можно подняться только
по винтовой лестнице.

Френсис Бэкон

К успеху придет только тот, кто способен преодолевать
неудачи не теряя энтузиазма.

Уинстон Черчилль

Об авторе

Стив Макконнелл – первое лицо Construx Software, где он возглавляет работы по инженерии ПО и ведет занятия в рамках программы профессионального развития фирмы. Стив пишет книги и статьи. Он автор «Code Complete» (1993 г.)[1], «Rapid Development» (1996 г.) и «Software Project Survival Guide» (1998 г.). Его книги были дважды удостоены премии Джолта (Jolt Excellence Award) журнала *Software Development* в номинации «Книга года в области разработки ПО». В 1998 г. читатели этого журнала назвали Стива Макконнелла одним из трех самых влиятельных людей в отрасли ПО наряду с Биллом Гейтсом и Линусом Торвальдом. В 1998–2002 г.г. Макконнелл являлся главным редактором журнала *IEEE Software*. Он вице-председатель Комитета профессиональных методик в Компьютерном обществе IEEE и входит в комитет экспертов проекта SWEBOK (Software Engineering Body of Knowledge, область знаний инженерии ПО).

Степень бакалавра Макконнелл получил в Колледже Уитмана, а степень магистра инженерии ПО – в Университете Сиэттла. Живет в Беллвью, штат Вашингтон.

Если у вас есть замечания или вопросы по этой книге, пишите Стиву Макконнеллу на *stevemcc@construx.com* или свяжитесь с ним через его сайт *www.stevemcconnell.com*.

---

[1] Стив Макконнелл «Совершенный код». – Пер. с англ. – СПб.: Питер, 2006.

Оглавление

Благодарности

Хотел бы поблагодарить многих специалистов, приславших свои замечания по ключевым разделам книги, среди которых Дон Багерт (Don Bagert), Джон Бентли (Jon Bentley), Стивен Блэк (Steven Black), Роберт Бернс (Robert C. Burns) из компании «Boeing», Тревор Берридж (Trevor Burridge), Аугусто Коппола (Augusto Coppola), Алан Корвин (Alan B. Corwin) из «Process Builder», Райан Флеминг (Ryan Fleming), Пэт Форман (Pat Forman), Роберт Гласс (Robert L. Glass) из «Computing Trends», Дэвид Гудман (David Goodman), Оуейн Гриффитс (Owain E. Griffiths), Брейди Хонсингер (Bradey Honsinger), Ларри Хьюз (Larry M. Hughes) из компании «Sprint», Роберт Ли (Robert E. Lee), Эйвонелл Ловхог (Avonelle Lovhaug), Марк Лутц (Mark Lutz), Стив Мэттингли (Steve Mattingly), Грант Мак-Лахлин (Grant McLaughlin), Брайан Мак-Лин (Brian P.McLean), Хэнк Мьюрет (Hank Meuret), Х. Фернандо Наведа (J. Fernando Naveda), Энтон Пэнг (Anthon Pang), Дэвид Парнас (David L. Parnas), Мэтт Пелоквин (Matt Peloquin), Том Рид (Tom Reed), Кейти Роуд (Kathy Rhode), Стив Ринн (Steve Rinn), У. Пол Роджерс (Wm. Paul Rogers), Джей Силвермен (Jay Silverman), Андре Синтцофф (Andrй Sintzoff), Тим Старри (Tim Starry), Стив Токи (Steve Tockey), Леонард Трипп (Leonard L. Tripp), Том Вентцер (Tom Ventser) из группы «DMR Consulting Group», Карл Вигерс (Karl Wiegers) и Грег Уилсон (Greg Wilson).

Мои благодарности также многочисленным рецензентам, высказавшимся по отдельным конкретным вопросам.

Особо хочу поблагодарить великолепный коллектив по подготовке книги издательства Addison-Wesley: Майка Хендриксона (Mike Hendrickson), Ребекку Гринберг (Rebecca Greenberg), Эйми Флейшер (Amy Fleischer), Кэрин Хансен (Karin Hansen) и Дженис Оуенс (Janis Owens). Рабо-

тать с каждым из них одно удовольствие, и книга стала значительно лучше в результате их труда.

Я также высоко ценю опыт сотрудничества при подготовке первого издания этой книги – «After the Gold Rush». Хотел бы напомнить об огромной работе редактора проекта Виктории Тульман (Victoria Thulman) и других сотрудников издательства: Бена Райана (Ben Ryan), Роба Нанса (Rob Nance), Черил Пеннер (Cheryl Penner) и Полы Горелик (Paula Gorelick).

Введение

Кажется, что это просто... пока не попробуешь.

Из журнала IEEE Software[1]

Я сидел в самолете, стоявшем на взлетной полосе, когда прозвучало объявление капитана: «У нас неполадки в системе кондиционирования самолета. Эта система поддерживает уровень кислорода на борту, поэтому она должна заработать раньше, чем мы взлетим. Перезапуск кондиционеров не удался, поэтому мы сейчас выключим и снова включим электропитание. *Знаете, все эти новые самолеты управляются компьютерами, поэтому они не слишком надежны*».

Пилот выключил и снова включил питание – по сути «перезагрузил» самолет, и рейс продолжился без происшествий. Нечего и говорить, что по окончании воздушного путешествия я с большой радостью вышел из самолета.

Лучшие времена и худшие

Лучшие разработчики ПО ведут свои проекты так, чтобы обеспечить достижение целевых показателей качества. Они точно планируют сроки сдачи ПО на месяцы и годы вперед. Проекты разработки ПО укладываются в выделенный бюджет, и производительность таких разработчиков постоянно растет. Моральный дух их персонала высок, и клиенты очень довольны.

---

[1] Из книжного обзора, посвященного [137].

- Телекоммуникационной компании понадобилось изменить около 3 тысяч строк в базовом ПО объемом примерно в 1 000 000 строк. Изменения были внесены столь тщательно, что через год работы не обнаружилось ни одной ошибки. Время, которое потребовалось для внесения изменений, включая анализ требований, планирование, реализацию и тестирование, составило *9 часов* [110].
- Группа разработчиков ПО для BBC США взялась реализовать некий проект за год с бюджетом $2 000 000, хотя другие вполне достойные разработчики предлагали срок до 2 лет при бюджете до $100 000 000. Когда же эта группа сдала ПО на месяц раньше срока, менеджер проекта заявил, что успех достигнут за счет методик, известных уже несколько лет, но редко применяемых на практике [49], [131].
- Авиастроительная компания разрабатывает ПО для клиентов по фиксированной цене, при этом только 3% ее проектов превышают сметную стоимость; 97% из 100 укладываются в бюджет.[1]
- Организация, твердо следующая политике достижения исключительного качества ПО, в течение 9 лет добивалась ежегодного снижения на 39% количества дефектов, обнаруживаемых после выпуска версий; итоговое снижение составило 99% [56].

Вместе с огромными успехами, примеры которых приведены выше, отрасль ПО приносит в экономику миллиарды долларов как за счет прямых продаж самого ПО, так и в результате повышения эффективности и производительности, а также создания продуктов и услуг, которые возможны только при использовании соответствующего ПО.

Методики, необходимые для создания качественного программного продукта, известны уже 10, а то и 20 лет. Тем не менее, несмотря на впечатляющие достижения, отрасль ПО не использует весь свой потенциал. Между передовыми разработчиками и общей массой существует огромный разрыв, а многие широко применяемые методики сильно устарели и не обеспечиваются достаточными ресурсами. Эффективность среднего проекта ПО оставляет желать лучшего, о чем свидетельствуют многие хорошо известные провалы.

- Налоговая служба США провалила программу модернизации ПО стоимостью $8 000 000 000, что обошлось в $50 000 000 000 несобранных доходов в год [3].

[1] Из разговора с автором.

- Улучшенная АСУ Федерального управления авиации (FAA) превысила выделенный бюджет примерно на $3 000 000 000 [17], [53], [48].
- Неполадки в системе обработки багажа привели к задержке открытия международного аэропорта в Денвере более чем на год. Потери оцениваются в $1 100 000 в день [48], [53].
- Ракета «Ариан-5» взорвалась при первом пуске из-за ошибки в ПО [99].
- Бомбардировщик «Б-2» также не взлетел с первого раза из-за проблем с ПО [44].
- Управляемые компьютером паромы в г. Сиэттл (штат Вашингтон) около полутора десятка раз врезались в доки, нанеся ущерб на сумму свыше $7 000 000. Власти штата рекомендовали выделить более $3 000 000 на перевод паромов обратно на ручное управление [98].

Подобным ошибкам подвержены и другие проекты. Около четверти из них терпят полную неудачу с самого начала [132], [66]. Очень часто к моменту сворачивания проекта выявляется двукратный перерасход бюджета. Примерно половина всех проектов либо затягивается, либо превышает сметную стоимость, либо обеспечивает меньше функциональных возможностей, чем предусматривалось [132].

Для предприятий такие свернутые проекты означают упущенные возможности: если бы закрытие проекта обходилось в 10% выделенных средств, а не в 200%, нетрудно представить, чего можно было бы добиться, просто перенаправив эти ресурсы в проекты, которые были завершены.

На национальном уровне отмененные проекты представляют собой чудовищную и бесполезную трату сил и ресурсов. Грубые подсчеты показывают, что свернутые проекты ПО обходятся экономике приблизительно в $40 000 000 000.[1]

---

[1] Этот грубый расчет основан на статистике занятости, представленной в табл. 7.2 «Структура занятости работников сферы ПО» по должностям ученых компьютерно-информационной отрасли, проводящих исследования; программистов-компьютерщиков; инженеров прикладного ПО; инженеров системного ПО и аналитиков компьютерных систем. Другие должности в этом анализе не учитывались. Итоговые расходы экономики США на разработку ПО рассчитывались путем умножения средних совокупных расходов на оплату труда ($95 000) на 1 741 000 работников этих должностей. Четверть итоговой суммы в $160 000 000 000 25% – это доля, затраченная на отмененные проекты. В этом анализе может не учитываться влияние отмененных проектов, поскольку риск отмены увеличивается с ростом объема проекта, поэтому отмененные проекты могут оказаться более дорогостоящими по сравнению со средними расчетами.

Но и успешные проекты могут представлять угрозу безопасности и общественному благополучию. Руководителю проекта в Lotus звонил хирург, который использовал электронную таблицу для анализа состояния пациента во время операции на открытом сердце [142]. В журнале «Ньюсвик» публиковались фотографии военных, планирующих военные операции при помощи Microsoft Excel на своих переносных компьютерах; группа технической поддержки Excel принимала телефонные звонки с поля боя во время активных военных действий.

Цель данной книги

Процесс разработки ПО можно сделать прогнозируемым, контролируемым, экономичным и управляемым. Обычно разработка ПО ведется иначе, однако есть все возможности делать именно так. Эта книга посвящена зарождающейся профессии – инженерии ПО (т. е. технологии разработки ПО) и практической профессиональной методологии, которая обеспечивает экономичное создание высококачественного программного обеспечения.

В книге обсуждаются следующие вопросы:

- Что такое инженерия разработки ПО?
- Как инженерия ПО связана с компьютерной наукой, т. е. с наукой о вычислительных системах?
- Почему обычного программирования недостаточно?
- Зачем нужна *профессия* инженерии разработки ПО?
- Почему *инженерия* является наилучшей моделью профессиональной разработки ПО?
- В чем отличия эффективных методик, используемых в различных проектах (или различных компаниях), и какие принципы практически одинаковы?
- Что могут предпринять организации, чтобы поддержать профессиональный подход к разработке ПО?
- Что нужно сделать индивидуальным разработчикам ПО, чтобы стать профессионалами?
- Что могут предпринять представители отрасли разработки ПО в целом, чтобы создать настоящую профессию «инженерия ПО»?

Структура книги

Материал книги постепенно переходит от рассмотрения программирования как ремесла в его сегодняшнем состоянии к изучению технологии ПО как профессии, которая может сформироваться в будущем.

Часть I «Смоляная яма программного обеспечения» содержит рассказ о том, как отрасль эволюционировала к своему нынешнему состоянию. Этот процесс определялся многими факторами, понимать которые необходимо, для того чтобы ускорять, а не замедлять наступление перемен, призванных сделать успешные проекты ПО повседневной реальностью.

Часть II «Индивидуальный профессионализм» рассматривает шаги, которые специалист может сделать самостоятельно для достижения высокого профессионального уровня в разработке ПО.

Проекты ПО настолько сложны, что многие ключевые факторы невозможно обсудить на уровне индивидуального разработчика. В части III «Организационный профессионализм» представлены организационные методики, необходимые для поддержания более высокого профессионализма в программных проектах. Часть IV «Индустриальный профессионализм» рассматривает меры, которые должны быть предприняты в масштабе отрасли, чтобы обеспечить профессиональный подход на индивидуальном и организационном уровнях.

У этой книги есть партнерский сайт, *www.construx.com/profession*, на котором размещены материалы, связанные с этой книгой, включая списки профессиональной литературы для чтения, планы самообучения, описание существующих инициатив по сертификации и лицензированию, а также ссылки на многие другие полезные сайты.

Что я узнал с 1999 г.

Книга «Профессиональная разработка программного обеспечения» представляет собой обновленный и значительно расширенный вариант моей книги, вышедшей в 1999 г. [86]. С 1999 г. я усвоил несколько положений, которые нашли отражение в моей новой книге.

- Вопрос лицензирования разработчиков ПО оказался более спорным, чем я ожидал. Я по-прежнему считаю, что лицензирование небольшого количества инженеров-программистов – это важная часть защиты жизнедеятельности людей и их безопасности. Я старался разъяснить, что лицензирование представляет собой всего лишь

одну из множества инициатив, направленных на повышение профессионализма разработчиков ПО, и не самая важная.

- Образовательную подготовку инженеров-программистов не обязательно жестко увязывать с лицензированием. Программы подготовки на младших и выпускных курсах могут быть направлены на формирование инженерного склада мышления у разработчиков ПО, но при этом не обязательно на их подготовку к получению лицензии профессионального инженера. Если лицензию в конечном итоге получат менее 5% разработчиков ПО, что кажется вполне вероятным, то ориентация большинства программ подготовки и обучения на получение обучающимися лицензии представляется неразумной.
- Мир не рухнул первого января 2000 г., когда считалась актуальной угроза масштабных сбоев в работе компьютерных систем (я не думал, что проблемы, связанные с наступлением 2000 г., будут катастрофическими). Мрачные прогнозы не подтвердились. Более того, сама проблема Y2K была в определенном смысле вызвана *успешной* технологией разработки ПО. Она не возникла бы, если бы столь многие системы ПО не просуществовали значительно дольше, чем предполагалось.
- Современные разработки ПО действительно во многом впечатляют, поэтому любые рассуждения о профессионализации отрасли должны учитывать ее успехи. Необходима чрезвычайная осторожность, чтобы в попытках усилить слабые стороны процесса не отказаться от проверенных и успешных методик.

Кому адресована эта книга

Тем, для кого *разработка ПО служит источником средств к существованию*, эта книга даст представление о шагах, которые следует предпринять, чтобы стать настоящим профессионалом в этой области.

Менеджеры проектов разработки ПО найдут здесь свод отличительных особенностей, по которым хорошо управляемые проекты ПО можно отличить от проектов, управляемых плохо, а также обзор методов, которые могут сделать проекты более успешными.

Руководителям организаций-разработчиков ПО эта книга укажет преимущества и выгоды системного подхода к разработке ПО, а также действия, необходимые для их реализации.

ГЛАВА ВТОРАЯ

Ложное золото

Надежда хороша на завтрак, но не годится на ужин.
Френсис Бэкон

Проблемы с ПО сохраняются частично в силу завораживающей притягательности нескольких неэффективных практических подходов. Во времена Калифорнийской «золотой лихорадки» в конце 40-х – начале 50-х годов XIX века некоторых золотоискателей обманывало «ложное золото» – пирит железа, который блестит и сияет, как настоящее золото. Однако, в отличие от настоящего, «ложное» золото – вещество хрупкое, слоистое, ломкое и почти ничего не стоит. Опытным золотодобытчикам хорошо известно, что настоящее золото мягкое, пластичное и не крошится под давлением. Уже 50 лет разработчики ПО поддаются соблазну своего «ложного золота». Негодные практические методы имеют соблазнительную привлекательность, но оказываются «ложным золотом», и, как и пирит железа, они хрупкие, ломкие и практически ничего не стоят.

Перемещение каменных глыб

Заглянем еще дальше в историю, на много веков, задолго до Калифорнийской «золотой лихорадки», и предположим, что вы строите одну из древних пирамид. Перед вами стоит задача: переместить огромную каменную глыбу на 10 км от реки на место строительства, как показано на рис. 2.1. У вас есть 20 человек и 100 дней, чтобы переместить этот камень.

Вам разрешено пользоваться любым методом, чтобы камень оказался на нужном месте. Нужно каждый день передвигать камень на 100 метров

к строящейся пирамиде, в противном случае придется изобрести что-нибудь, сокращающее срок, необходимый для преодоления оставшегося расстояния.

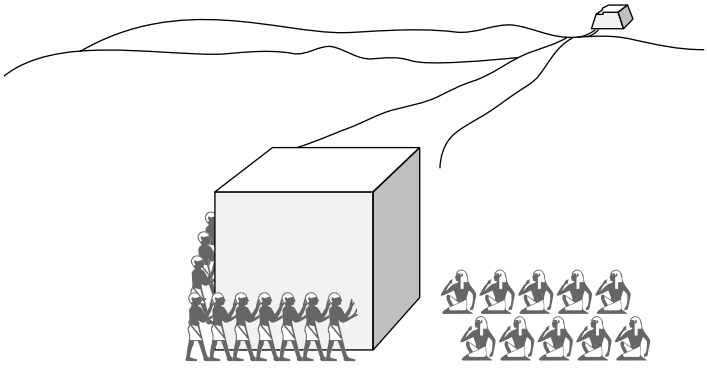

Рис. 2.1. *На проект разработки ПО можно смотреть как на перемещение тяжелой каменной глыбы. Нужно либо передвигать камень каждый день ближе к конечному пункту, либо попытаться придумать нечто, позволяющее сократить на один день срок, необходимый для преодоления оставшегося расстояния*

Некоторые бригады «передвижников» сразу бы взялись толкать камень, прилагая грубую силу. Этот способ мог бы быть эффективным в случае небольшого камня. Но если крупная каменная глыба покоится на песчаной поверхности пустыни, ее вряд ли удастся передвигать таким способом более-менее быстро, если она вообще поддастся усилиям рабочих. Если глыба перемещается на десять метров в день, то можно считать удовлетворительным результатом, что она вообще приближается к конечному пункту, но при этом каждый день бригада отстает на 90 метров. «Продвижение» не всегда означает достаточный прогресс.

Бригада рабочих пособразительней не стала бы сразу приниматься за толкание глыбы. Лишь очень небольшие камни поддаются воздействию грубой силы. Если камень крупный, надо потратить некоторое время на планирование перемещения и только потом прикладывать мускульные усилия. Подумав над задачей, такие рабочие могли бы спилить несколько деревьев и использовать их как катки (рис. 2.2). На это ушло бы день-два, но очень возможно, что приспособление ускорило бы передвижение глыбы.

Но если деревьев рядом нет и надо потратить несколько дней, чтобы поискать их вдоль реки? Вероятно, прогулка вдоль реки – штука достаточ-

но полезная, поскольку бригада, намеревающаяся применить грубую силу, сможет передвигать глыбу лишь на малую часть того расстояния, которое требуется преодолевать каждый день.

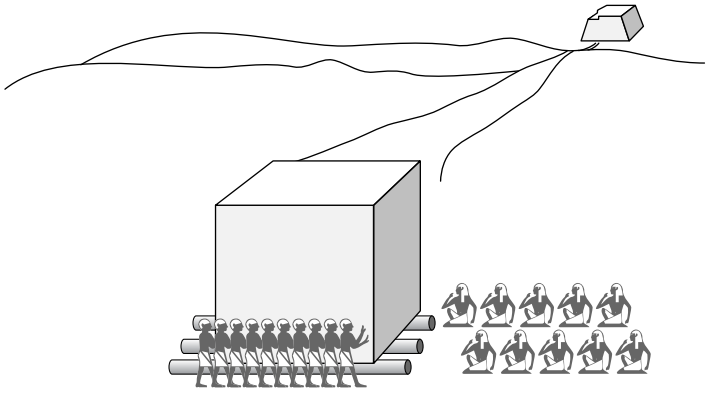

Рис. 2.2. *Передвигая каменную глыбу или создавая компьютерное ПО, рассудительные работники спланируют работу, чтобы она шла споро и эффективно*

Рассуждая аналогичным образом, можно также предположить, что сообразительная бригада решит каким-то образом подготовить путь, по которому она будет перемещать глыбу. Эти рабочие не будут катить груз по песку, а проложат дорогу, что будет особенно полезно, если надо переместить не одну, а несколько глыб.

По-настоящему изобретательная бригада, начав с катков и подготовки дороги, в конечном итоге сообразит, что если в качестве катков используется лишь минимальное количество деревьев, то приходится слишком часто останавливаться, чтобы перенести вперед задний каток, освобождающийся при продвижении глыбы. Если запастись несколькими лишними катками и выделить рабочих для их переноски, то бригада сможет лучше поддерживать скорость движения.

Они также могут понять, что количество рабочих, которые могут разместиться у основания глыбы, чтобы толкать ее, ограничено. Поэтому можно тянуть глыбу при помощи лямок, одновременно толкая ее, как показано на рис. 2.3. Поскольку усилие распределяется на большее количество рабочих, нагрузка на каждого из них уменьшается, и быстрый шаг на самом деле оказывается более приемлемым, чем медленный.

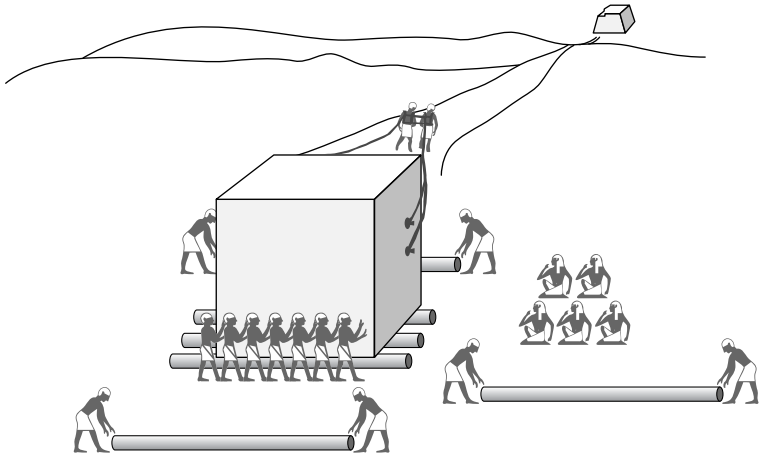

Рис. 2.3. *Творческие коллективы постоянно ищут пути повышения эффективности работы*

Каменные глыбы и программное обеспечение

Каким образом связаны каменные глыбы и ПО? Перемещение каменной глыбы сродни написанию исходного кода программы. Если на завершение проекта ПО отведено 100 дней, то надо либо писать одну сотую программы ежедневно, либо предпринять что-то, что позволит ускорить написание оставшейся части программы. Поскольку написание программ значительно менее осязаемо, чем перемещение каменной глыбы, продвижение в начале проекта разработки ПО оценить труднее. Для проектов ПО характерен «синдром последней минуты»: программисты слабо чувствуют срочность работы в начале проекта, без толку теряя целые дни, и вынуждены отчаянно торопиться в конце. Представив себе написание исходной программы как перемещение каменной глыбы, нетрудно понять, что нельзя успешно выполнить проект за счет отчаянного финишного рывка. Каждый день менеджер проекта должен спрашивать: «Приблизилась ли сегодня глыба на один день к конечному пункту? Если нет, то сократился ли объем работ на один день?»

Еще один аспект, в котором усматривается связь между передвижением глыбы и разработкой ПО, состоит в том, что в конечном итоге, сколько бы вы ни планировали работу, вы должны передвигать глыбу, то есть писать программу. Написание исходного кода во всех проектах, за исключением

совсем крошечных, предусматривает проработку множества деталей, и этот факт легко недооценить.

Сначала напишем, потом исправим ошибки

Из всех ошибок, допускаемых при разработке ПО, самая распространенная – это, безусловно, недостаточное внимание, уделяемое изготовлению катков и подготовке дороги. Почти 75% коллективов разработчиков начинают работу, налетая на глыбу и пытаясь подвинуть ее при помощи грубой силы.[1] Такой подход – сразу хвататься за написание программ, не позаботившись о планировании и проектировании – получил наименование «напишем и исправим». Иногда к этому подходу прибегают из-за того, что у программистов руки чешутся немедленно начать программировать. Иногда же дело в том, что руководству или заказчикам не терпится увидеть реальные проявления прогресса. Разработка по принципу «напишем и исправим» неэффективна во всех проектах, за исключением самых мелких.

Этот подход, как и попытка передвинуть глыбу грубой силой, нехорош тем, что быстрый старт не обязательно переходит в быстрое продвижение к финишу. Коллективы, практикующие более сложные подходы, формируют структуру, позволяющую проекту выйти на высокий уровень производительности и эффективно завершить его. Подкладывая катки под глыбу, расчищая дорогу и подготавливая четкое целенаправленное приложение усилий работников, можно создать такую структуру. Подход по принципу «напишем и исправим» подразумевает быстрое начало движения глыбы, но не обеспечивает ее достаточное перемещение каждый день, а приложение грубой силы не дает приемлемого результата. Обычно это приводит к сотням и тысячам ошибок уже на ранних этапах проекта. По результатам некоторых исследований, от 40 до 80% бюджета типичного проекта ПО расходуется на исправление дефектов, которые появились в нем ранее [13], [44], [63], [91], [140].

На рис. 2.4 показано, как постепенно снижается производительность в проекте, реализуемом по принципу «напишем и исправим». В начале

[1] Этот средний процент основывается на количестве проектов ПО в соответствии с моделью зрелости процессов создания ПО (Capability Maturity Model for Software – SW-CMM) уровня 1. В главе 14 эти статистические данные обсуждаются более подробно.

проекта прилагаются небольшие усилия (или вообще не прилагаются) в части планирования и управления процессом. Небольшую часть составляют непроизводительные затраты («пробуксовка»), но основное время посвящено программированию. По мере продвижения проекта к концу исправление дефектов становится все более существенной частью.

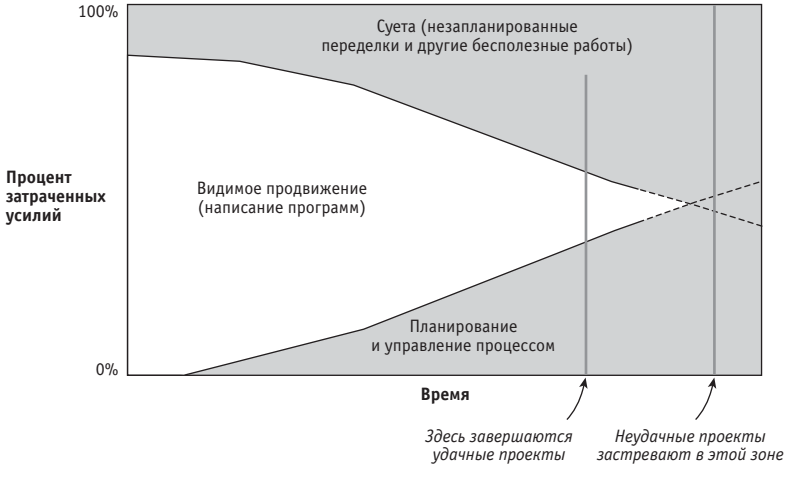

Рис. 2.4. *При использовании подхода «напишем и исправим» удачные проекты завершаются, когда от них еще можно получить небольшие «порции» производительной работы. Неудачные проекты застревают в области, где все 100% усилий направлены на переработку, планирование и управление процессом. Источник: [84]*

Согласно рис. 2.4 удачные проекты, выполняемые по принципу «напишем и исправим» доводятся до конца, но процесс выдачи небольших порций программы продолжается. Неудачные проекты застревают на стадии, когда 100% усилий уходят на планирование, управление процессом и бесполезную переделку, а программирование не продвигается. Если усилия по планированию не были предприняты заранее, то написанные программы рассыпаются вдребезги. Некоторые из таких проектов еще можно спасти, если подтолкнуть коллектив разработчиков достаточно далеко (влево по диаграмме), чтобы удалось выдать приемлемый продукт. Остальные проекты в конечном итоге сворачиваются.

Эта печальная картина вовсе не преувеличение. В нескольких исследованиях утверждается, что около 25% всех проектов разработки ПО в конечном итоге просто отменяются [62], [64], [132]. На момент сворачивания

такой проект наверняка превысил выделенный бюджет, а его участники погрязли в бесконечном отыскивании ошибок, тестировании и переделках. Причины закрытия проекта в том, что проблемы с качеством кажутся непреодолимыми [64].

Ирония этой ситуации заключается в том, что при проведении этих неудачных проектов в конечном итоге приходится затрачивать столько же усилий на планирование процесса и управление им, сколько и в удачных проектах. Приходится организовывать процесс отслеживания дефектов, чтобы исправить все обнаруженные ошибки. По мере приближения даты сдачи ПО проводится более тщательная оценка. К концу проекта коллектив разработчиков может проводить переоценки чуть ли не каждую неделю или даже каждый день. Определенные усилия и время тратятся на то, чтобы убедить заказчиков и заинтересованные стороны в том, что ПО в конце концов будет сдано. Не исключается отслеживание дефектов и внедрение стандартов отладки участков программы до их окончательной интеграции с уже отлаженными фрагментами. Но поскольку такая практика часто реализуется на поздних стадиях проекта, ее преимущества удается использовать лишь в небольшой части всей работы.

Тем не менее методы, применяемые в неудачных проектах, отличаются от тех, что были бы выбраны при более эффективной организации на ранних стадиях проекта. А многие практические меры, предпринимаемые в первом случае, оказываются ненужными, если управление проектом с самого начала организовано более разумно.

Как показывает рис. 2.5, самые продвинутые фирмы – те, что производят самые надежные программные продукты при наименьшей стоимости и в кратчайшие сроки – тратят относительно небольшую часть бюджета на стадии непосредственного создания ПО. Те, кто демонстрирует наименьшую изобретательность, тратят почти весь свой бюджет на программирование и исправление ошибок в программах. Поэтому совокупные бюджеты таких коллективов значительно выше, поскольку не закладывается основа для эффективной работы. (В главе 14 мы вернемся к этому вопросу.)

Разработка по принципу «напишем и исправим» продолжает применяться, потому что такой подход привлекателен по двум причинам. Во-первых, это позволяет коллективу разработчиков немедленно увидеть продвижение: можно начать с передвижения глыбы на 10 метров в первый день, пока более разумный коллектив пилит деревья для катков, ровняет дорогу для надежного движения и еще не сделал никаких видимых дости-

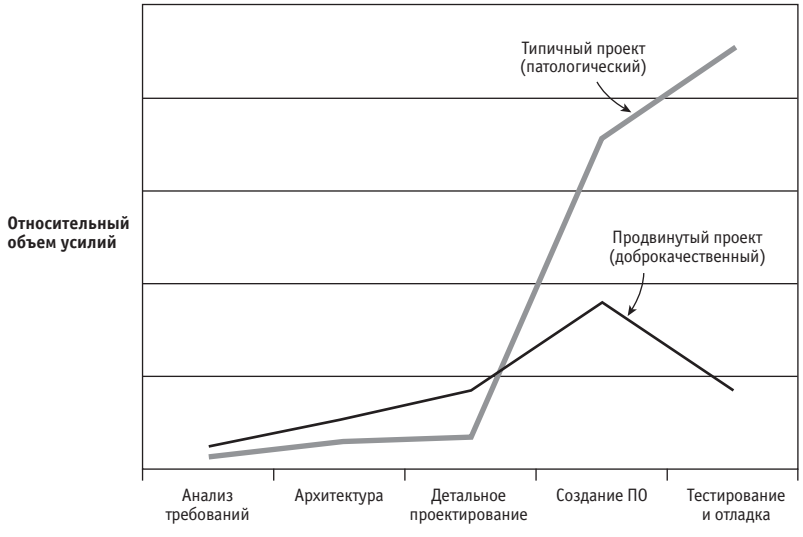

Рис. 2.5. *Развитые методики разработки ПО предусматривают больше усилий на ранних стадиях проекта с целью избавиться от избыточных работ на более поздних стадиях*[1]

жений в перемещении глыбы. Если менеджеры и заказчики не слишком разбираются в сложностях динамики успешного проекта, то подход «напишем и исправим» выглядит очень привлекательным. Второй аспект привлекательности этого подхода состоит в том, что не требуется никакого обучения. В отрасли, где средний уровень подготовленности в проектировании ПО достаточно невысок, этот метод принимался как самый распространенный по умолчанию.

Принцип «напишем и исправим» – одна из форм «ложного золота» в ПО. На первый взгляд он кажется заманчивым, но опытные разработчики ПО понимают его бесперспективность.

[1] Характеристика «развитого проекта» основана на работах, выполненных Лабораторией проектирования ПО НАСА. Описание «типичного проекта» составлено на основе данных о проектах, собранных автором во время его работы в качестве консультанта, и соотносятся с данными из работы [66] и из других источников.

Ориентир – качество

Можно предположить, что сроки выполнения проекта ПО удастся сократить, если тратить меньше времени на тестирование или технические экспертизы. Приверженцы принципа «напишем и исправим» в создании ПО утверждают: «Накладные расходы бесполезны». Отраслевой опыт свидетельствует об обратном. Попытка разменять качество на расходы или сроки реально приводит к увеличению расходов и удлинению сроков.

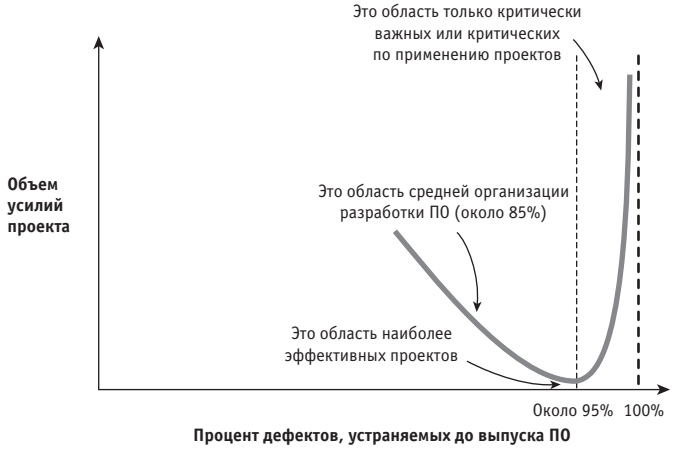

Рис. 2.6. *До определенного момента проекты, в которых добиваются наименьшего числа дефектов, также могут уложиться в самые короткие сроки. В большинстве проектов можно сократить сроки, устраняя дефекты на более ранних стадиях работы. Данные взяты из [67]*

Как показано на рис. 2.6, проекты, в которых примерно 95% дефектов удается устранить до выпуска ПО, оказываются наиболее продуктивными и меньше всего времени при их реализации тратится на исправление собственных ошибок. В таких проектах следует приложить больше усилий для повышения качества. Там, где такие показатели не достигаются, можно повысить эффективность за счет устранения большего количества дефектов на ранней стадии разработки. В эту категорию в настоящее время попадают примерно 75% проектов ПО. Для них попытка обменять качество на снижение расходов и сокращение сроков является еще одним примером «ложного золота». Это также можно рассматривать как пример уже известной динамики развития проектов ПО. Фирма IBM еще лет 25 назад обнару-

жила, что в проектах, концентрирующих свои усилия на минимизации сроков, часто превышаются и сроки, и расходы. А там, где усилия фокусируются на минимизации количества дефектов, достигаются кратчайшие сроки и наивысшая производительность работ [63].

Иногда «ложное золото» оказывается серебром

Технологии и методологии, которые увязываются с утверждениями о сногсшибательной производительности, часто называют «серебряными пулями», поскольку предполагается, что они сразят низкую производительность, как серебряная пуля бьет наповал оборотней [20]. Десятилетиями отрасль ПО бомбардировали заявлениями, что технология «фикс» кардинально ускоряет процесс разработки. В 60-х годах прошлого века это было программирование в диалоговом режиме, затем в 70-х годах его сменили языки программирования третьего поколения, а в 80-е годы подобные обещания давали проповедники искусственного интеллекта и инструментария CASE. В 90-е годы объектно-ориентированное программирование преподносилось как очередная панацея. А в начале XXI века на эту роль была выдвинута интернет-разработка ПО.

Предположим, что попытки «передвинуть каменную глыбу» начинаются с применения силы. Через несколько дней руководитель группы начинает понимать, что дело идет слишком медленно и выполнить задачи проекта не удастся. К счастью, ему приходилось слышать о чудесном животном, именуемом «слон». Слоны достигают веса, в 100 раз превосходящего вес взрослого человека, и обладают огромной силой. Руководитель группы снаряжает экспедицию, поставив перед ней задачу отловить слона. После трехнедельного сафари экспедиция приводит слона. Могучее животное запрягают в лямку, чтобы оно тянуло глыбу, и щелкают хлыстом. Работники, затаив дыхание, следят, насколько быстро слон сможет тащить глыбу. Может быть, удастся доставить глыбу к месту строительства даже раньше запланированного срока! Под их напряженными взглядами слон начинает тянуть глыбу вперед намного быстрее, чем это когда-либо удавалось людям. Но внезапно слон встает на дыбы, рвет упряжь и, раздавив двух работников, убегает со скоростью 40 км/час, только его и видели (рис. 2.7). Люди в отчаянии. Им приходит в голову, что, наверное, надо было научиться управлять слоном, прежде чем начинать реальную работу с его участием. Тем временем на поиски слона затрачено более 20% выделенного на проект времени, два их товарища погибли, а приблизиться к цели не удалось.

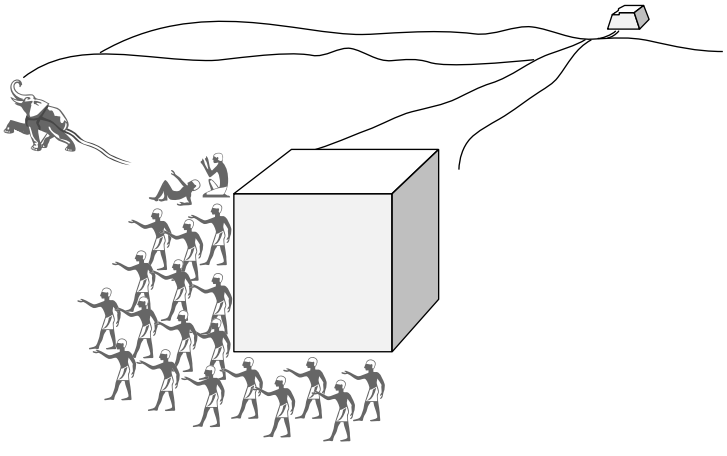

Рис. 2.7. *Инновации типа «серебряной пули» часто не оправдывают ожиданий*

Это и есть суть синдрома «волшебной серебряной пули».

Аналогия со слоном точнее, чем может показаться. В работе [53] Роберт Л. Гласс (Robert L. Glass) описал 16 проблемных проектов разработки ПО. От четырех из описанных им проектов ожидался прорыв в достижении успеха, поскольку применялись инновационные технологии типа «серебряной пули». Вместо ошеломляющего успеха последовал провал, причем именно из-за этих инноваций.

Особый тип «серебряной пули» получается в результате небрежной реализации методик повышения эффективности организационных процессов. В некоторых фирмах организационные инновации пытаются внедрить с помощью слов-заклинаний, таких как комплексное управление качеством (Total Quality management – TQM), технология развертывания функций качества (Quality Function Deployment – QFD), модель зрелости процессов создания ПО (Capability Maturity Model for Software – SW-CMM), «нулевой дефект», «шесть сигма», «непрерывное совершенствование», «статистический контроль процессов». Все эти методики полезны при надлежащем применении и практическом внедрении по сути, а не просто по форме. Если же им отвести роль заклинаний, все они станут никчемными. В некоторых фирмах за год успевают пройти полный цикл их применения, как будто ритуальные заклинания и повторение магических слов, свидетельствующие об увлечении менеджментом, могут повысить качество и

производительность. Фирмам, где придерживаются такой практики, в аду низкой производительности отведено особое место. После нескольких лет управления посредством заклинаний сотрудники относятся презрительно к любым организационным инновациям в целом, что усложняет отказ от разработки по принципу «напишем и исправим».

Правильно выбранные инновационные методы в подходящем проекте, сопровождаемые соответствующим обучением и внедряемые с учетом реалистичных ожиданий, могут дать огромные выгоды, если им придается статус долгосрочной стратегии. Но последние нововведения не обладают волшебными свойствами, и их не так легко внедрить. Если от таких нововведений ждут мгновенной выгоды, то они становятся «ложным золотом».

Программное обеспечение – это не пластилин

Еще один вид «ложного золота» – это убеждение, что ПО представляет собой «мягкий» объект, в отличие от оборудования, которое изменить трудно. Изначально ПО было названо «мягким» (software), потому что его было легко изменить. Для очень небольших программ на заре программирования это, возможно, так и было. Но по мере усложнения систем ПО представление о программах как о легко изменяемом объекте стало одним из самых пагубных заблуждений в их разработке.

Согласно некоторым исследованиям, изменение технических требований (попытка воспользоваться предполагаемой «мягкостью» ПО) – это самый распространенный или один из самых распространенных источников превышения бюджета и сроков [64], [79], [132], [138] и один из важнейших факторов отмены проектов. В некоторых случаях изменения в результате «ползучести» требований могут настолько «раскачать» продукт, что его невозможно завершить вообще [64].

Приведем простой пример, доказывающий, что ПО не так легко изменить, как представляется многим. Пусть требуется спроектировать систему, которая первоначально будет распечатывать набор из пяти отчетов, а в конечном итоге должна выдавать 10 отчетов. Есть несколько аспектов гибкости («мягкости»), которые должны быть приняты во внимание:

- Число десять – это абсолютный предел количества выдаваемых отчетов?
- Будут ли последующие отчеты аналогичны первым пяти?
- Всегда ли будут распечатываться все отчеты?
- Всегда ли отчеты будут распечатываться в одном и том же порядке?

- До какой степени пользователь сможет настраивать для себя отчеты?
- Сможет ли пользователь формировать свои собственные отчеты?
- Можно ли будет настраивать и задавать отчеты по ходу работы?
- Будут ли отчеты переводиться на другие языки?

Как бы тщательно ни разрабатывалось ПО, всегда будет существовать точка, где ПО уже нельзя будет легко изменить. В рассматриваемом случае системы отчетов любое из следующих требований может оказаться жестким:

- Задание более десяти отчетов.
- Задание нового отчета, отличного от начального набора из пяти отчетов.
- Распечатка подмножества отчетов.
- Распечатка отчетов в задаваемом пользователем порядке.
- Возможность для пользователя настраивать отчеты.
- Возможность для пользователя формировать полностью собственный отчет.
- Перевод отчетов на другой язык, основанный на латинском алфавите.
- Перевод отчетов на другой язык, основанный на нелатинском алфавите, или язык с порядком чтения справа налево.

Этот пример любопытен тем, что автор мог бы задать множество вопросов о «гибкости» этих отчетов, не зная ровным счетом ничего конкретного о них или даже о системе, в которой они будут распечатываться. Зная просто о существовании «неких отчетов», можно поднять много общих вопросов о различной степени гибкости системы.

Было бы соблазнительно утверждать, что разработчики должны всегда предусматривать максимальную гибкость своего ПО, но гибкость может меняться почти бесконечно и обходится в немалые суммы. Если пользователю надо, чтобы пять отформатированных отчетов всегда печатались вместе, в одном порядке и на одном языке, то разработчику не следует конструировать развитое программное средство (утилиту) для создания настраиваемых отчетов. Такой продукт может легко оказаться в 100–1 000 раз дороже, чем обеспечение основных простых функциональных возможностей, реально необходимых пользователю. Пользователь же (или клиент, менеджер) должен оказать помощь разработчикам ПО при определении требуемой адаптируемости.

За гибкость надо платить в данный конкретный момент. Ее ограничение сэкономит средства именно в данный момент, но если ее понадобится

обеспечить в будущем, то она обойдется дороже в разы. Трудность инженерного решения заключается в соотнесении известных на данный момент требований и возможных будущих потребностей и определении степени гибкости или жесткости при создании продукта.

К каким выводам приводит существование «ложного золота»

В заключение приведем несколько «мягких» утверждений, которые можно было бы назвать самоочевидными (или же которые становятся очевидными при ближайшем рассмотрении):

- Для успеха проекта ПО нельзя начинать написание исходной программы на слишком ранней стадии.
- Нельзя жертвовать контролем количества дефектов ради стоимости или сроков проекта, если только речь не идет о критически важных системах. Держите количество дефектов в центре внимания; стоимость и сроки приложатся.
- Магические спасительные средства типа «волшебной серебряной пули» вредны для проекта, хотя практика показывает, что их поставщики будут утверждать обратное.
- Небрежная модификация технологии является особенно губительной «серебряной пулей», поскольку подрывает дальнейшие попытки усовершенствовать процесс разработки.
- Несмотря на представление о ПО как о гибком предмете, оно таковым не является, если только не было изначально разработано гибким, а создание гибкого ПО стоит дорого.

Чтобы извлечь эти уроки, у мира программного обеспечения было 50 лет. Наиболее успешные личности и организации восприняли их очень серьезно. Научиться постоянно сопротивляться соблазну «ложного золота» – это один из первых шагов, которые должна сделать отрасль ПО на пути к формированию настоящей профессии разработчика ПО.

ГЛАВА ЧЕТВЕРТАЯ

Разработка ПО – это не компьютерная наука

Ученый создает, чтобы научиться;
инженер учится, чтобы создавать.

Фред Брукс

Один из моих любимых вопросов во время собеседований с претендентами на должность программиста следующий: «Как бы вы описали ваш подход к программированию?» Я приводил в качестве примера плотника, пожарника, архитектора, художника, писателя, ученого, изыскателя и археолога и предлагал дать кандидатам их собственный ответ. Некоторые из них пытались предугадать, что я хотел услышать от них; обычно они отвечали, что видят себя учеными. «Крутые» программисты отвечали, что ощущают себя членами ударной группы или отряда десантников. Мне больше всего понравился такой ответ: «При проектировании ПО я архитектор. Когда я конструирую интерфейс пользователя, я художник. Когда я пишу ПО, я ремесленник. А когда я тестирую программу, я ужасная сволочь».

Мне нравится задавать этот вопрос, потому что он ставит на повестку дня фундаментальный вопрос: как лучше всего рассматривать разработку ПО? Это наука? Искусство? Ремесло? Или же нечто абсолютно иное?

«Есть» и «должно быть»

Спор о том, что такое разработка ПО – наука или искусство, идет уже давно. Тридцать лет назад Дональд Кнут (Donald Knuth) взялся за напи-

сание семитомного «Искусства программирования». Первые три тома содержали 2200 страниц, так что можно предположить, что во всех семи томах могло бы быть более 5 тысяч страниц. Если это *искусство* программирования, то не уверен, что мне когда-нибудь захочется взглянуть на *науку*.

Те, кто считает программирование искусством, указывают на эстетические аспекты разработки ПО и утверждают, что наука не допускает такого воображения и творческой свободы, а те, кто считает программирование наукой, указывают на огромное количество ошибок в программах, утверждая, что столь низкая надежность недопустима, и черт с ней с творческой свободой. Обе эти точки зрения грешат неполнотой и ставят во главу угла неверный тезис. Разработка ПО – это искусство, наука, ремесло, археология, тушение пожаров, социология и еще много других видов деятельности человека, взятые вместе. В некоторых областях это любительство, в других – профессионализм. Это столько же различных вещей одновременно, сколько существует программистов. Но правильно поставленный вопрос состоит не в том, что такое есть разработка ПО на данный момент, а скорее в том, чем должна быть профессиональная разработка ПО. С моей точки зрения, ответ на этот вопрос ясен: профессиональная разработка ПО должна быть инженерией. Такова ли она сегодня? Нет. А должна быть? Несомненно.

Инженерия и наука

Лишь около 40% разработчиков ПО имеют диплом специалиста по вычислительным системам (computering science)[1], и практически никто из них не имеет диплома по разработке ПО, поэтому не стоит удивляться путанице между первой и второй формулировками. Различие между наукой и инженерией в программном обеспечении такое же, как и в других областях знаний.[2] И ученые, и инженеры познают истину. Ученые проверяют гипотезы и расширяют знания в своей предметной области. Инженеры отыскивают полезные знания и учатся применять полученные знания для решения практических задач. Ученые должны быть в курсе новейших исследований. Инженерам должны быть известны методы, надежность и эффективность которых уже подтверждена. В занятиях наукой позволительна узкая специализация. Инженер должен понимать и учитывать все

[1] Именно так называется степень выпускника зарубежных университетов. – *Примеч. науч. ред.*

[2] Большей частью этих размышлений я обязан Дэвиду Л. Парнасу (David L. Parnas), особенно его работе [108].

факторы, влияющие на разрабатываемое изделие. За учеными не нужен надзор, потому что об их работе судят главным образом другие ученые. Инженерам необходимы нормативы, поскольку результаты их работы используют обычные люди. Научное образование готовит выпускников к продолжению учебы. Инженерное образование готовит студентов к практической работе сразу после завершения учебы.

Университеты выдают дипломы по специальности «вычислительные системы», и обычно считается, что студенты, получившие эту специальность, найдут работу как разработчики ПО и сразу начнут решать задачи реального мира. Лишь небольшая часть студентов, получивших специальность «вычислительные системы», продолжает обучение или идет в научно-исследовательские организации, где развиваются знания о программном обеспечении или компьютерах.

Таким образом, эти выпускники оказываются на «нейтральной технологической территории». Их именуют учеными, но они выполняют функции, которые традиционно входили в обязанности инженеров, и при этом не имеют инженерной подготовки. Такой же результат можно получить, поручив ученому-физику конструировать бытовую электротехнику. Физик, возможно, глубже понимает научные принципы электричества, чем инженер-электрик. Но его опыт создания аппаратуры сводится к изготовлению прототипов, используемых в лаборатории для расширения научных знаний. У него нет опыта или просто даже необходимой подготовки для создания надежного экономичного оборудования, которое решает практические задачи в условиях реального мира. Можно рассчитывать, что оборудование, сконструированное кандидатом наук, физиком, будет работать, но у него не будет той надежности, которая делает прибор практичным или безопасным за пределами лаборатории. В его оборудовании материалы могут использоваться допустимым лишь для единственного прототипа образом, что чрезвычайно расточительно при изготовлении крупных партий изделия.

В области ПО ситуации, подобные приведенному выше примеру с физиком, исчисляются буквально тысячами каждый год. Когда сотрудники, имеющие специальность ученого по вычислительным системам, приступают к разработке производственных систем, они часто проектируют и создают ПО, которое либо слишком слабо для производственных целей, либо небезопасно. Они глубоко и узко сосредоточены на мелких задачах и игнорируют более важные факторы. Они могут потратить два дня на шлифовку вручную какого-нибудь алгоритма сортировки, вместо того чтобы за два

часа позаимствовать его из библиотеки стандартных подпрограмм или найти в подходящей книжке. Типичному выпускнику по специальности вычислительных систем нужно несколько лет практической деятельности, чтобы набрать достаточное количество практических знаний для создания минимально приемлемого производственного ПО без надзора специалиста. Не имея формального образования, некоторые разработчики ПО, посвящают данной области всю жизнь, так и не обретя этих знаний.

Недостаток профессионализма – это беда не только разработчиков ПО: вся сфера ПО стала жертвой собственного успеха. Рынок труда в области ПО растет быстрее, чем образовательная инфраструктура, необходимая для поддержания его роста, поэтому больше половины специалистов, занимающих должности разработчиков ПО, получили образование по другим специальностям. Работодатели не могут требовать от этих «обращенных в ПО» сотрудников, чтобы они получали необходимое инженерное образование в свободное от работы время. Но если бы работодатели и могли это сделать, то все равно большинство курсов переподготовки посвящены науке в области вычислительных систем, а не разработке ПО. Образовательная инфраструктура отстает от нужд отрасли.

Что стоит за модным словечком?

Некоторые эксперты считают, что «инженерия ПО» (software engineering) – это лишь очередное модное словосочетание, аналогичное по значению программированию компьютеров. Очевидно, что термин «инженерия ПО» употребляется неверно, но имеет все же вполне конкретное значение.

Толковые словари определяют слово «инженерия» как применение научных и математических принципов в практических целях. Именно это и пытается делать большинство программистов. Они берут научно разработанные и математически определенные алгоритмы, методы функционального конструирования, методики обеспечения качества и другие методики и разрабатывают программные продукты. Как указывает Дэвид Парнас (David Parnas), профессия инженера давно получила законный статус, поэтому потребители знают, кто имеет квалификацию, позволяющую создавать технические продукты [107]. Потребители программного обеспечения заслуживают того же отношения.

Существует, однако, мнение, что, считая разработку ПО инженерной профессией, мы должны применять формальные методы – создавать про-

граммы как строгие математические доказательства. Опыт и здравый смысл подсказывают, что одного этого более чем достаточно для гибели многих проектов. Приверженцы другого взгляда возражают, что коммерческое ПО слишком сильно зависит от условий рынка, чтобы разработчики могли позволить себе роскошь тщательного продолжительного инженерного конструирования.

Такие возражения основаны на узком и неверном понимании смысла инженерии. Ведь инженерия – это применение научных принципов в *практических* целях. Если инженерия непрактична, то это плохая инженерия. Попытка применить формальные методы ко всем проектам создания ПО так же порочна, как и разработка всего и вся по принципу «напишем и исправим».

Подход к созданию ПО как к предмету инженерии делает понятнее тезис о том, что проекты ПО дифференцируются в зависимости от их целей. Материалы для проектируемого здания подбираются такие, чтобы они соответствовали его назначению. Большой ангар-укрытие для хранения сельскохозяйственного инвентаря и механизмов можно построить из листов тонкого железа без теплоизоляции. Строить из них жилой дом вряд ли разумно. Но, хотя дом прочнее и теплее, нельзя утверждать, что укрытие хуже дома: оно спроектировано в соответствии с предполагаемым назначением. Если бы укрытие для инвентаря было спроектировано подобно жилому дому, можно было бы даже критиковать такое «избыточное проектирование» за бесполезно потраченные ресурсы для строительства и смело назвать его плохим.

В области ПО правильно организованный проект может быть ориентирован на достижение любой из следующих целей создания программного продукта:

- Минимизация числа дефектов
- Повышение степени удовлетворенности пользователя
- Сокращение времени отклика
- Удобство обслуживания
- Хорошая масштабируемость
- Высокая степень устойчивости
- Высокая степень корректности[1]

[1] И это еще далеко не полный перечень критериев. – *Примеч. науч. ред.*

Каждая проектная команда должна четко определить относительную важность этих характеристик, а затем уже осуществлять проект в соответствии с задачей их достижения.

Проекты по разработке ПО отличаются от инженерных проектов, в которых используются физические материалы. В других инженерных областях стоимость материалов может составлять до 50% стоимости всего проекта. Многие производственные фирмы утверждают, что проекты, в которых оплата труда превышает 50% их стоимости, автоматически получают статус высокого риска [5]. В типичном проекте по разработке ПО стоимость труда может доходить до почти 100% всего бюджета. Многие инженерные проекты направлены на оптимизацию свойств создаваемого продукта; стоимость проектирования при этом относительно невелика. Поскольку оплата труда составляет столь высокую долю всех расходов на протяжении жизненного цикла ПО, программные проекты должны быть ориентированы на достижение целей проекта в большей степени, чем другие инженерные проекты. Поэтому помимо обеспечения характеристик продукта коллективу разработчиков ПО, возможно, придется работать на достижение следующих целей проекта:

- Короткие сроки
- Прогнозируемая дата сдачи продукта
- Низкие затраты
- Реализация проекта силами ограниченного коллектива разработчиков
- Обеспечение гибкости внесения изменений в требования по ходу проекта

В каждом проекте должен соблюдаться баланс между характеристиками продукта и задачей создания продукта в целом. Никто не хочет платить $5000 за текстовый редактор, но никому не нужен и редактор, который будет зависать каждые 15 минут.

Конкретные характеристики продукта и проекта, поставленные во главу угла проектной командой, сами по себе не определяют данную работу как «настоящий» проект разработки ПО. Иногда требуется выдать продукт с минимумом дефектов и почти безукоризненной надежностью – это ПО для медицинской аппаратуры, авионики, тормозных систем и т. д. Трудно не согласиться, что такие разработки образуют подходящую сферу для полноценных инженерных проектов ПО. В других проектах ставится цель обеспечить продукт с достаточной надежностью, но при минимальных за-

тратах и в кратчайшие сроки. Являются ли такие проекты подходящей сферой приложения для инженерии ПО? Одно неформальное определение инжиниринга – это «выполнение за 10 центов работы, которую каждый может сделать за доллар». Во многих проектах ПО сегодня за доллар делается то, что любой хороший разработчик ПО мог бы сделать за 10 центов. Экономичная разработка – это тоже сфера приложения инженерии проектов ПО.

Охвативший сегодня почти всю отрасль ПО принцип разработки «написать и исправить» и сопутствующее ему превышение бюджета и сроков есть не результат самого инженерного подхода, а следствие недостатка специальной подготовки и образования в сфере инженерии ПО.

Правильные вопросы

Обычная практика разработки ПО сегодня не очень похожа на инженерию, но она вполне может быть таковой. Как только мы перестанем задавать неверный вопрос: «*Что такое* разработка ПО сегодня?» и начнем задавать правильный: «*Должна ли* разработка ПО быть инженерной практикой?», мы начнем отвечать на действительно интересные вопросы. Что является ядром области знаний инженерии ПО? Что нужно сделать, чтобы профессиональные разработчики ПО могли воспользоваться этими знаниями? Насколько велика отдача от практики разработки ПО как инженерной дисциплины? Каковы стандарты профессионального поведения разработчиков ПО, организаций-разработчиков ПО? Надо ли регулировать деятельность разработчиков ПО? Если да, то до какой степени? И возможно, самый интересный вопрос из всех: какой будет отрасль разработки ПО, когда на все эти вопросы будут получены ответы?