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.

Студенты, которые хотели бы работать в области создания ПО, познакомятся здесь с основами технологии ПО и получат представление о карьере в этой отрасли.

На пути к профессиональной разработке ПО

Исследователи, специализирующиеся в этой области, давно указывали, что производительность компаний, конкурирующих в одной отрасли, может отличаться более чем в десять раз [91]. Последние исследования показали, что разница может достигать и 600 раз [145]. Самые эффективные компании действительно процветают.

Выгоды формирования настоящей профессии разработчика ПО весьма убедительны. Согласно традиционным воззрениям, наибольший риск представляют перемены. Если же говорить о программном обеспечении, то наибольший риск представляет как раз их отсутствие и погружение в болото порочных, экстравагантных технологий разработки ПО вместо принятия методов, давно уже доказавших свою практическую эффективность.

Как реализовать такой переход? Это и есть главная тема книги.

– Беллвью, штат Вашингтон.
День Поминовения, 2003 г.

ЧАСТЬ ПЕРВАЯ

Смоляная яма программного обеспечения

ГЛАВА ПЕРВАЯ

Динозавры в смоляной яме

Тот, кто не хочет прибегать к новым средствам, должен ожидать новых бед.

Френсис Бэкон

Фредерик Брукс еще в 1975 г. сравнил разработку крупных систем программного обеспечения с борьбой динозавров, мамонтов и саблезубых тигров с засасывающей липкой смолой в яме [21], предсказав на долгие годы вперед сохранение подобной ситуации в проектировании и разработке ПО, когда одну ногу можно освободить, только завязнув другой ногой.

Проблемы, которые Брукс выявил почти тридцать лет назад, уже тогда не были новы, так что у разработчиков ПО было более четверти века, чтобы поработать над ними. Так далеко ли удалось продвинуться с тех пор?

Многие проблемы, буквально преследующие почти любой проект разработки ПО, практически не изменились. К примеру, вопрос сроков выполнения остается типичным и для проектов сегодняшних дней. По некоторым оценкам примерно 75% средних и свыше 90% крупных проектов ПО сталкиваются с проблемами жестких сроков.[1] При этом сверхурочные переработки программистов являются скорее правилом, чем исключением.[2] Хорошо известно, что современные зарождающиеся предприятия, работающие в области ПО, ожидают от своих сотрудников значительной сверхурочной работы, а рассказы о программистах, засыпающих ночью

---

[1] Данные о статистике напряженности в связи со сроками выполнения взяты из работы [64]. Данные о типичном соблюдении сроков сдачи проектов ПО имеются в работах [64], [66], [132].

[2] Этот вопрос подробно обсуждается в книге автора [83].

за своими дисплеями, можно услышать почти всюду [18]. Однако еще в середине 60-х годов XX века в одном из исследований утверждалось, что «многие программисты, не укладываясь в сжатые сроки, просиживают ночи на рабочих местах» [22]. В 1975 г. Фредерик Брукс указывал, что «из-за нарушения сроков исполнения срывается куда больше проектов, чем в силу всех остальных причин вместе взятых» [19]. Так что превышение сроков исполнения проектов наблюдается уже более тридцати лет, а если учесть, что люди нетерпеливы по природе, то и значительно дольше.

Размах сегодняшних проектов ПО кажется беспрецедентным, что, разумеется, наталкивает их создателей на мысль, что никто и никогда не сталкивался с проблемами подобного масштаба. Тем не менее даже у гигантских проектов, вроде Windows NT, были предшественники. Объемы современных крупных проектов разработки ПО действительно впечатляют, причем каждый разработчик естественно полагает, что он первый берется за столь масштабный проект. Однако, как я уже сказал, прецеденты существовали и ранее. Например, первоначальный проект Windows NT потребовал около полутора тысяч человеко-лет трудозатрат,[1] однако разработка OS/360 фирмы IBM, завершенная в 1966 г., оказалась в три раза более трудоемкой.[2]

Данные последних исследований показывают, что наиболее часто встречающиеся причины неудач проектов по разработке ПО связаны с проблемами их соответствия заданным техническим условиям, которые выстраивают неверную систему. Эти требования слишком размыты, чтобы их можно было точно реализовать, а некоторые из них меняются настолько часто, что вносят полнейшую неразбериху в системный проект [24], [132]. Но все же проблемы с соответствием требованиям также далеко не новы. Еще в 1969 г. Роберт Фрош (Robert Frosch) заметил, что система «может удовлетворять букве технических условий, но тем не менее не являться целиком и полностью удовлетворительной» [47].

Большинство разработчиков ломают голову, пытаясь поспеть за стремительными изменениями, которые происходят благодаря развитию Интернета. Как угнаться за новыми языками, меняющимися стандартами и предложениями новых программных продуктов? Для тех, кто работает

---

[1] Это примерная сумма. Оценка основана на опубликованных данных о стоимости разработки Windows NT ($150 000 000) и совокупных расходах на оплату труда в $100 000 на человека в год [146].

[2] Трудозатраты составили примерно 5000 человеко-месяцев [19].

в сфере программирования уже пару десятков лет, нынешняя ситуация кажется очень похожей на ту, что была в середине 80-х годов прошлого века, когда пришествие персональных компьютеров *IBM* полностью изменило использование вычислительных машин в компаниях.

Во времена разработки языка программирования FORTRAN в 1954–1958 гг. предполагалось, что он устранит сложность программирования компьютеров: ученые и инженеры просто вводили бы свои формулы в компьютер, а тот транслировал бы их в машинный код; отсюда и название FORTRAN – FORmula TRANslation (ТРАНслятор ФОРмул). Но, разумеется, FORTRAN не исключил программирование, а лишь сократил его объем на машинном языке. Время от времени появляются многообещающие заявления о возможности автоматизации программирования [118]. Компьютеры станут настолько «умными», что нужда в программистах и вовсе отпадет. Однако эта пластинка уже была заезжена более 35 лет назад, когда Джин Билински подметил, что «описание бизнесмена, бодро общающегося со своим всемогущим компьютером на обычном языке, регулярно появляется в прессе» [22]. Реальность же заключается в том, что подробное до мелочей описание проблемы – задача весьма трудная, и эта сторона программирования никуда не уйдет. Новые инструменты полезны, но они не заменяют ясность мышления. Я написал об этом в 1996 г. в своей книге «Rapid Development» [83], однако Роберт Фрош уже утверждал то же самое в издании *IEEE Spectrum* 30 лет назад.

Если говорить о разработке ПО в эпоху Интернета., то он дает возможность его создателям легко распространять обновленные версии своих программ. Пользователи могут загружать обновления программ в электронном виде без необходимости копирования их на компакт-диски CD или DVD – доставка быстро и недорого. В свою очередь, это увеличивает давление на разработчиков, чтобы те чаще выпускали обновления программ в ответ на требования пользователей. Интернет-разработчики утверждают, что пользователи просто хотят скорее получить ПО, нежели обрести его безукоризненную версию. У интернет-разработчиков в ходу поговорка: «Лучше быть первым, чем правым».

Насколько действительно беспрецедентна эта ситуация? Некоторые интернет-разработчики считают, что такая динамичность уникальна именно для веб-проектов, но ветеранам отрасли программирования известно: низкая стоимость распространения ПО, легкость внесения исправлений, невысокие потери в случае неудач – все это сильно смахивает на произ-

водственную среду разработки ПО своими силами для компьютеров коллективного пользования.

Эти тенденции, сложившиеся за 25 лет истории разработки ПО, являются источником и радости, и понимания того факта, что некоторые проблемы существуют уже четверть века и достаточно распространены. Мы действительно увязли в этой смоляной яме слишком давно, но повод для оптимизма все же есть: достаточно долго имея дело с одними и теми же проблемами, мы можем понять их сущность, и, как я намерен показать в этой книге, похоже, что мы находимся на пути их решения.

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.

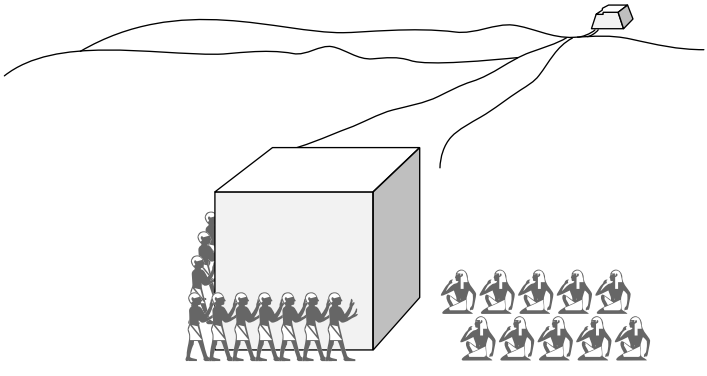

Рис. 2.1. *На проект разработки ПО можно смотреть как на перемещение тяжелой каменной глыбы. Нужно либо передвигать камень каждый день ближе к конечному пункту, либо попытаться придумать нечто, позволяющее сократить на один день срок, необходимый для преодоления оставшегося расстояния*

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.

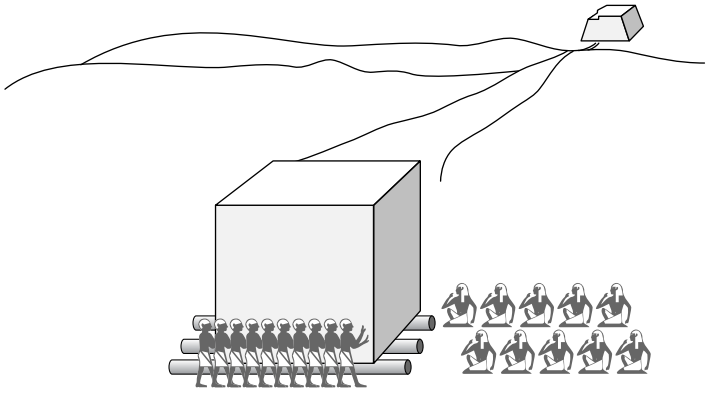

Рис. 2.2. *Передвигая каменную глыбу или создавая компьютерное ПО, рассудительные работники спланируют работу, чтобы она шла споро и эффективно*

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.

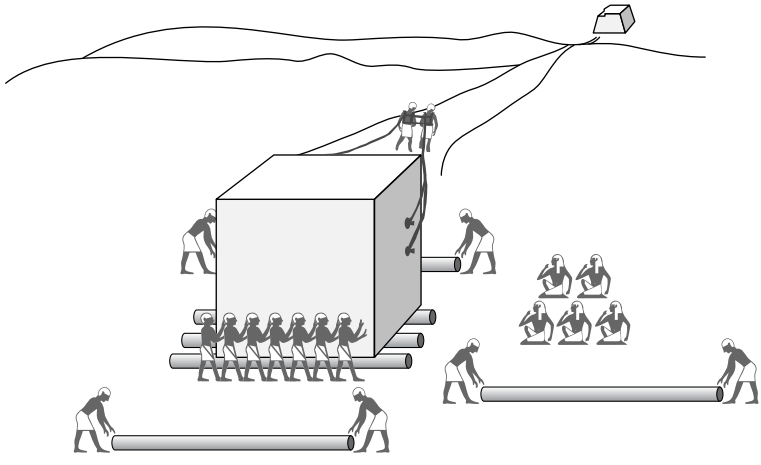

Рис. 2.3. *Творческие коллективы постоянно ищут пути повышения эффективности работы*

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.

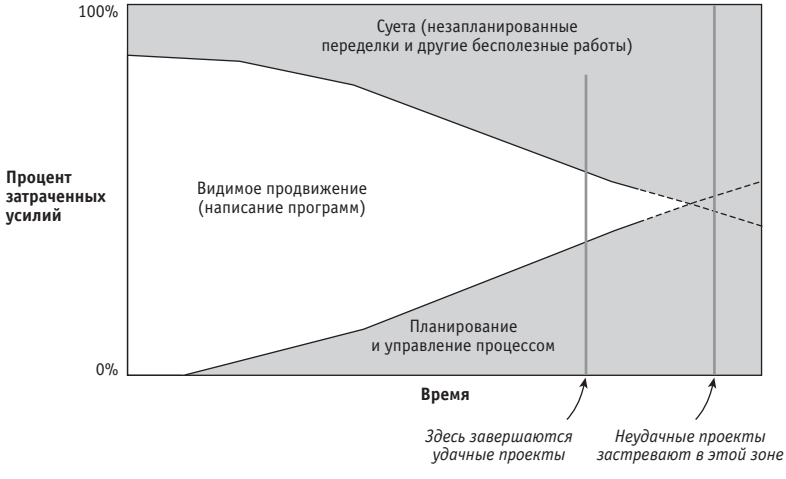

Рис. 2.4. *При использовании подхода «напишем и исправим» удачные проекты завершаются, когда от них еще можно получить небольшие «порции» производительной работы. Неудачные проекты застревают в области, где все 100% усилий направлены на переработку, планирование и управление процессом. Источник: [84]*

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-

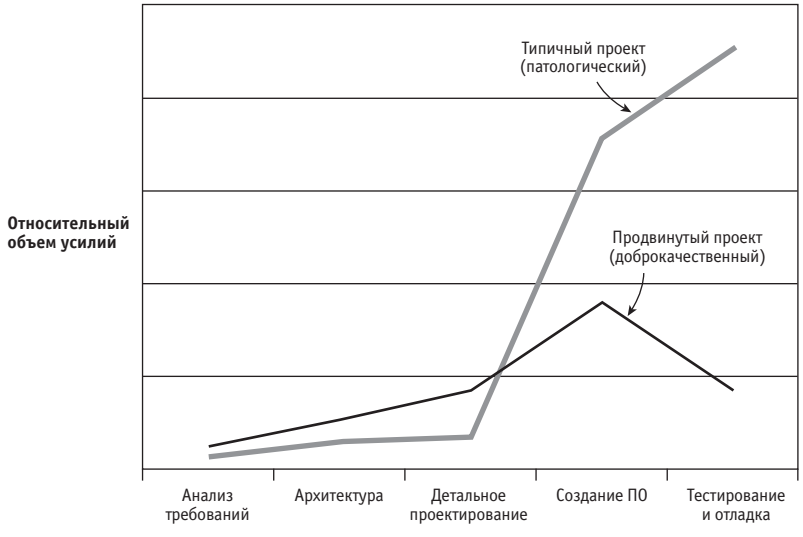

Рис. 2.5. *Развитые методики разработки ПО предусматривают больше усилий на ранних стадиях проекта с целью избавиться от избыточных работ на более поздних стадиях*[1]

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.

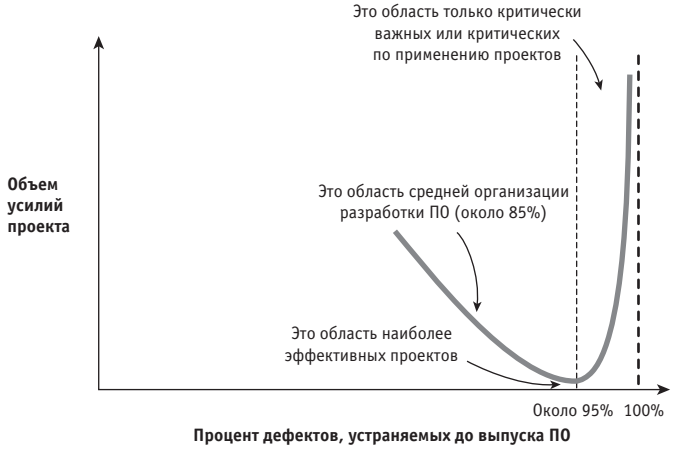

Рис. 2.6. *До определенного момента проекты, в которых добиваются наименьшего числа дефектов, также могут уложиться в самые короткие сроки. В большинстве проектов можно сократить сроки, устраняя дефекты на более ранних стадиях работы. Данные взяты из [67]*

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.

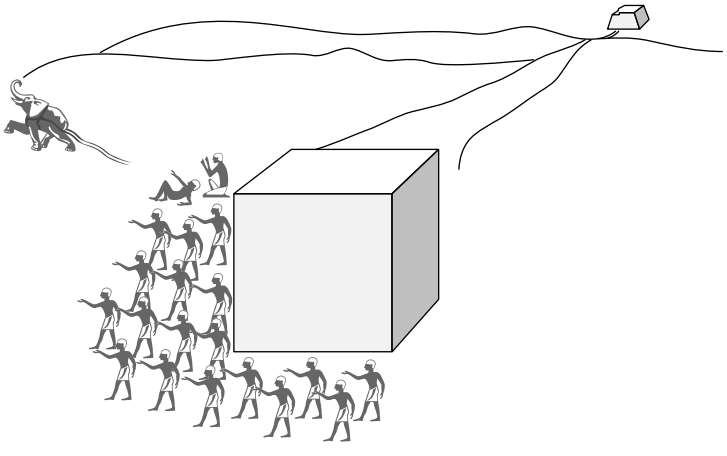

Рис. 2.7. *Инновации типа «серебряной пули» часто не оправдывают ожиданий*

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 ТРЕТЬЯ

«Культ карго» в разработке ПО

У народностей, населяющих регионы южных морей, бытует «культ карго». В войну к ним прилетали самолеты с массой полезных вещей. Теперь люди хотят, чтобы так было опять. Поэтому они устраивают некое подобие взлетно-посадочной полосы, вдоль нее разжигают костры, строят будку, в которой сидит человек, изображающий диспетчера (с деревяшками вместо наушников и бамбуковыми палочками-антеннами), и ждут приземления самолета. Они все делают как нужно. По форме все правильно. Все выглядит так, как было раньше. Вот только самолеты не приземляются. Я называю такие вещи наукой «культа карго»: соблюдаются все внешние признаки и рецепты научного исследования, но нет чего-то очень важного, потому что самолеты так и не приземляются.

Ричард Фейнман,
лауреат Нобелевской премии по физике [61]

Полезно различать два разных стиля разработки ПО, а именно: стиль, ориентированный на процесс, и стиль, ориентированный на энтузиазм (героизм) личности. Первый направлен на достижение результата посредством тщательного планирования детально прописанных процессов, эффективного использования выделенного времени и продуманного внедрения лучших методов разработки ПО. Этот подход ведет к успеху, поскольку его применение заставляет фирму совершенствовать свои методы. Даже если первые попытки не слишком эффективны, постоянное внимание к процессу разработки означает, что каждая последующая попытка будет успешнее предыдущей.

Второй стиль разработки известен под несколькими названиями. Организации, отдающие предпочтение этому стилю разработки ПО, отличают-

ся тем, что привлекают лучших в своей области разработчиков, требуют от них полной отдачи и концентрации на проекте, обеспечивая им почти полную автономию и мотивируя их в наивысшей степени, а потом смотрят, чтобы они работали по 10, 12 или 16 часов в день, пока проект не будет завершен. Сила личностно-ориентированного стиля заключается в его огромных мотивационных возможностях, поскольку самые разные опросы и исследования постоянно показывают, что индивидуальная мотивация намного опережает все остальные факторы по вкладу в производительность [11]. Разработчики добровольно принимают личные обязательства и часто идут на огромные жертвы ради успеха проектов, в которых они заняты.

Самозванцы от ПО

При разумном применении каждый из этих двух подходов к разработке ПО может привести к созданию высококачественного программного продукта в кратчайшие сроки и с наименьшими затратами. Но у каждого из них есть патологические двойники, не сравнимые с ними по результативности, но которые бывает трудно отличить от оригинала.

Главная черта самозванцев-двойников первого стиля заключается в бездумном копировании процесса ради самого процесса. Эти организации пытаются быть похожими на такие процессно-ориентированные структуры, как Лаборатория NASA по разработке ПО или бывшее подразделение Федеральных систем фирмы IBM. В этих структурах вырабатывается много документов, и часто проводятся совещания. Отсюда делается вывод, что если плодить эквивалентное количество документов и проводить сравнимое число совещаний, то можно добиваться аналогичных успехов. А если выдавать еще больше документации и созывать еще больше совещаний, то можно и превзойти их успех. Отсутствует понимание, что не документация и совещания обеспечивают успех, что все это просто побочные эффекты нескольких чрезвычайно эффективных процедур. Организации, в которых получили распространение такие заблуждения, мы называем *бюрократическими* – в них форма процесса разработки ПО превалирует над содержанием. Подобное извращенное понимание процессов снижает мотивацию, а с ней и производительность разработчиков. Да и работать в таких организациях не слишком-то приятно.

Фирмы, в которых приживается «героический» стиль, главным образом сконцентрированы на том, чтобы стимулировать сотрудников работать как можно больше времени. Наблюдая за практикой фирмы Microsoft, они

видят, что вырабатывается очень мало документации, сотрудникам предлагаются возможности покупки акций фирмы, а взамен требуется много отрабатывать сверхурочно. Отсюда делается вывод, что если тоже минимизировать объем документации, предлагать различные возможности и требовать огромных переработок, то успех придет сам собой. И чем меньше документации и больше переработки, тем лучше. Однако упускается из виду тот факт, что Microsoft и другие «ориентированные на героизм» фирмы *не требуют* работать сверхурочно. Они берут на работу программистов, которые обожают создавать программы. Эти компании объединяют таких людей в команды с им подобными, обеспечивая щедрую организационную поддержку и вознаграждения. После этого они дают им полную свободу. Естественный результат – разработчики и менеджеры добровольно предпочитают работать допоздна. Следствие (продолжительность работы) при этом подменяется причиной (высокой мотивированностью). Организацию труда в таких фирмах можно назвать *потогонной*, потому что упор делается не на продуктивный труд, а на интенсивный, который к тому же характеризуются хаотичностью и неэффективностью. Ну и работать в таких фирмах тоже не очень приятно.

«Культ карго» в разработке ПО

На первый взгляд эти два типа самозванцев противоположны друг другу. Первые страшно бюрократизированы, вторые олицетворяют собой хаос. Однако есть одна общая ключевая черта, которая на самом деле важнее, чем все их различия. Ни те, ни другие не могут быть эффективны, потому что ни те, ни другие не понимают, что же действительно делает проект успешным. Они стараются имитировать действия, чтобы быть похожими на эффективные фирмы, исповедующие тот же стиль. Но поскольку они не понимают причин эффективности методов, то втыкают бамбуковые палочки в уши и надеются на удачное приземление самолета. Многие проекты в таких фирмах проваливаются, потому что это просто две различные вариации «культа карго» в разработке ПО, похожие в своем непонимании причин успеха.

«Культ карго» в разработке ПО легко узнаваем. Инженеры-программисты, приверженцы этого культа, оправдывают свои методы работы словами «мы всегда так работаем» или «стандарты работы нашей компании требуют, чтобы мы работали так», даже если конкретные приемы просто бессмысленны. Они отказываются признать наличие компромиссов обоих

стилей, имеющих как недостатки, так и достоинства. Столкнувшись с новыми более удачными методиками, программисты «культа карго» предпочитают укрыться в своих «деревянных будках» – знакомых и удобных, но не обязательно эффективных, привычных методах. Старинная мудрость гласит, что если одно и то же дело делать одним и тем же способом, то ожидать различных результатов не приходится. (А вы, друзья, как ни садитесь...©) Это тоже проявление «культа карго» в сфере ПО.

Суть спора

Знатоки программирования часто спорят, что лучше: приверженность процессу или ориентация на «личный энтузиазм». Но это ложная альтернатива. Оба стиля хороши и могут сосуществовать.[1] Там, где обычно работают, ориентируясь на процесс, в отдельных проектах может потребоваться максимальная самоотдача, а там, где принят стиль «героический», можно увидеть примеры искусного применения методики, ориентированной на процесс.

Разница между двумя подходами фактически сводится к стилевым и личностным особенностям. Мне приходилось сталкиваться в ряде проектов и с тем и с другим подходами, и в каждом из них я находил для себя определенные преимущества. Некоторым программистам нравится методично работать по 8 часов в день 5 дней в неделю, что более характерно для компаний, где ориентируются на процесс. Для других характерна целеустремленность, и они получают удовольствие от почти непрерывной работы. В среднем атмосфера проектов, ориентированных на энтузиазм, более приподнятая, но и процессно-ориентированный проект может вдохновлять не меньше, если есть четко поставленная цель. Фирмы, в которых придерживаются процессно-ориентированного стиля, видимо, чаще вырождаются в своих патологических двойников, чем их «героические»

---

[1] Так, для известной методологии IBM Rational Unified Process (RUP) были разработаны «каркасы» (frameworks), описания процессов разработки, начиная от «экстремального программирования» – представителя «гибких (личностно-ориентированных) технологий» – до процессно-ориентированных. При этом, основываясь на опыте успешных проектов разработки ПО, первые рекомендуются для небольших проектов, а последние – для крупных проектов с большой проектной командой. (См. Якобсон А. (Jacobson Ivar), Буч Г. (Grady Booch), Рамбо Дж. (Rumbaugh James) «Унифицированный процесс разработки программного обеспечения» (Addison-Wesley, 1999). – Пер с англ. – СПб.: Питер, 2002).

собратья, но любой из двух стилей эффективен, если проект четко спланирован и реализуется подготовленными людьми.

Наличие патологических двойников у обоих стилей также не добавляет ясности в спор о том, какой из них лучше. У каждого стиля есть проекты удачные и проекты, завершившиеся провалом. Приверженцам процессно-ориентированного стиля это позволяет указывать на успешные проекты, выполненные благодаря ему, и на неудачные проекты, в которых большее значение придавалось энтузиазму работников. То же самое, с точностью до наоборот, делают и приверженцы второго стиля.

Суть вопроса, которая осталась в стороне спора о преимуществах двух стилей, настолько очевидна, что, возможно, осталась незамеченной, как пропавшее письмо в известном рассказе Эдгара По. Вопрос не в стиле разработки ПО, а в компетентности. Реальная разница не в том, какой стиль выбран, а в том, каким уровнем образования, квалификации и понимания обладают участники проекта. Вместо того чтобы спорить о достоинствах стилей, надо искать пути повышения среднего уровня разработчиков и компетентности менеджеров. Это увеличит шансы на успех проекта независимо от того, какой избран стиль.

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?

ГЛАВА ПЯТАЯ

Объем знаний

Истина скорее родится из ошибки, чем из путаницы.
Френсис Бэкон

Чтобы стать высококлассным специалистом в какой-либо области, человек должен освоить примерно 50 тысяч единиц информации, то есть блоков знаний, которые скорее можно запомнить, чем всякий раз воссоздавать [124]. В устоявшихся областях знаний, как правило, необходимо примерно 10 лет для накопления такого объема информации. Утверждают, что пакет знаний, необходимый для разработки ПО, недостаточно стабилен, чтобы о нем можно было говорить как о чем-то строго определенном. Говорят, половина того, что сегодня должен знать разработчик для создания ПО, устареет уже через три года. Если оценка «периода полураспада» правильна, то за 10 лет, необходимых для освоения 50 тысяч единиц информации, делающих человека специалистом мирового класса, 30 тысяч из них безнадежно устареют. Будущие разработчики ПО были бы похожи на Сизифа, толкающего камень на вершину горы, чтобы тот снова скатывался вниз, едва достигнув вершины.

Каков же «период полураспада» таких программных средств, как Java, Perl, C++, ОС Linux и Windows? Все эти средства весьма актуальны на момент написания данной книги, но останутся ли они таковыми в момент прочтения ее читателем? Принцип «периода полураспада» может вполне оказаться верным для знаний, относящихся к технологиям. Но есть и другой тип знаний, который послужит разработчику ПО на протяжении всей его карьеры, и эти знания не подвержены такому влиянию времени.

Суть и случайность

В 1987 г. Фредерик Брукс опубликовал очень важную статью [20]. Основное ее положение сводилось к тому, что никакая уникальная методология или инструментарий, никакая «серебряная пуля» не гарантирует десятикратное повышение производительности в течение последующих десяти лет. Обоснования этого тезиса помогают выявить знания в разработке ПО, которые не подпадают под принцип трехлетнего «периода полураспада».

Употребляя слова «сущность» и «случайность», Брукс опирался на древнюю философскую традицию различать «существенные» и «случайные» свойства или признаки вещей. Существенными свойствами вещь должна непременно обладать, чтобы быть собой. У автомашины должен быть двигатель, колеса, трансмиссия, чтобы она была автомашиной. Эти свойства существенные. Двигатель может быть V-образный или линейный, шины могут быть шипованные зимние или гоночные с гладким протектором, коробка передач – автоматическая или ручная рычажная. Эти свойства «случайные», они возникают по случаю и не влияют на базовую «автомобильность» машины. Слово «случайный» может сбивать с толку, но применительно к тезису Брукса означает «несущественный», «необязательный».

По Бруксу, основная трудность разработки ПО не в последовательном изложении понятий на конкретном компьютерном языке программирования (составление программы) или проверке точности этого представления (тестирование). Все это суть случайная часть разработки ПО. Сущность же его, по утверждению Ф. Брукса, состоит в выработке спецификаций, структуры и сверке строго определенного и высоко детализированного комплекса взаимоограничивающих положений. Он говорит, что программное обеспечение – вещь трудная в силу присущих ему сложности, изменчивости, неосязаемости и в силу того, что оно должно соответствовать принятым в отрасли стандартам.

Компьютерные программы *сложны* по своей природе. Если даже изобрести язык программирования, оперирующий понятиями на уровне области задачи,[1] программирование все же останется сложной работой, поскольку придется точно определять взаимосвязи между объектами реаль-

[1] Чистым и негипотетическим образцом такого средства программирования являются 5 языков визуального программирования автоматных схем, получившие широкое распространение в несколько последних лет в области построения систем АСУТП. Их использование как раз со всей отчетливостью демонстрирует тезис автора. – *Примеч. науч. ред.*

ного мира, устанавливать исключения, предусматривать все возможные переходы между состояниями и т. д. Если абстрагироваться от случайной работы, связанной с представлением этих факторов на конкретном языке программирования и в конкретной компьютерной среде, то придется столкнуться с сущностной трудностью определения основополагающих понятий реального мира и устранением ошибок в их осмыслении.

Другая большая трудность обусловлена необходимостью *соответствия* ПО многочисленным ограничениям реального мира, таким как уже существующее аппаратное обеспечение, компоненты третьих сторон, нормативы регулирующих органов, бизнес-правила и традиционные форматы данных. Проектировщики ПО часто сталкиваются с жесткими внешними факторами, ограничивающими потенциальную степень снижения сложности.

Еще одну существенную трудность представляет *изменчивость* ПО. Чем удачнее программа, тем больше для нее найдется применений и тем шире она будет адаптироваться за пределами той области, для которой она изначально предназначалась. По мере расширения ПО еще больше усложняется и должно соответствовать дополнительным требованиям. Чем больше адаптируется ПО, тем сильнее становятся адаптации.

Последняя трудность связана с присущим ПО недостатком *наглядности*. ПО невозможно визуально представить с помощью двух- или трехмерных геометрических моделей. Попытки визуально представить даже простейшую логику быстро становятся безнадежно сложными, что может подтвердить любой, кто когда-нибудь пытался составить блок-схему даже простой программы.

Брукс утверждает, что в сфере разработки ПО уже достигнуты все возможные улучшения случайных характеристик. Эти достижения включают изобретение языков высокого уровня, принятие интерактивных вычислений и развитие мощных интегрированных сред разработки. Ф. Брукс говорит, что дальнейшего повышения производительности разработки можно достичь только за счет преодоления существенных трудностей, присущих ПО: сложности, изменчивости, отсутствия наглядности и ограничений, накладываемых принятыми стандартами.

Формирование устойчивого ядра

Знания, которые помогают разработчикам преодолеть то, что Брукс называет «существенными трудностями» разработки ПО, мне видятся как

«принципы инженерии ПО», образующие ее ядро знаний. В 1968 г. НАТО провело первую конференцию по инженерии ПО. Употребление этого термина применительно к объему знаний того времени было преждевременным и имело целью спровоцировать дискуссию.

Но насколько малым было это самое устойчивое ядро в 1968 г.? Примем во внимание, что первый полностью строгий алгоритм бинарного поиска был опубликован в 1962 г., лишь за шесть лет до проведения упомянутой конференции НАТО [72]. К. Бом (C. B□hm) и Дж. Джакопини (G. Jacopini) опубликовали работу [15], формулирующую теоретическое обоснование исключения оператора перехода goto и предлагавшую структурное программирование лишь за два года до конференции. Эдсгер Дейкстра (Edsger Dijkstra) написал свое знаменитое письмо к редактору «О вреде оператора GOTO» в 1968 г. [40]. В то время идея применения подпрограмм была относительно свежей, а среди программистов постоянно шли споры о реальной полезности подпрограмм. Ларри Константин (Larry L. Constantine), Гленфорд Майерс (Glenford Myers) и Уэйн Стивенс (Wayne Stevens) опубликовали свою первую работу по структурному проектированию уже через шесть лет после конференции, в 1974 г. [25]. Том Гилб (Tom Gilb) опубликовал свою первую работу по метрикам ПО в 1977 г. [50], а первая книга Тома Демарко (Tom DeMarco) по анализу требований ПО вышла в 1979 г. [35]. Любой, кто взялся бы сформировать устойчивое ядро знаний в далеком 1968 г., получил бы разрозненные фрагменты работы.

Анализируя сферу области знаний в проекте SWEBOK (которая обсуждается в этой главе ниже), я оцениваю «период полураспада» области знаний инженерии ПО в 1968 г. лишь в 10 лет. Как видно из рис. 5.1, устойчивое ядро было относительно небольшим, и, по моим оценкам, всего от 10 до 20% знаний инженерии ПО 1968 г. еще применяется сегодня.

С 1968 г. инженерия ПО добилась существенного прогресса. Тысячи страниц написаны на эту тему. Профессиональные ассоциации проводят сотни конференций и рабочих семинаров ежегодно. Накопленные знания кодифицированы на более чем двух тысячах страниц стандартов IEEE в инженерии ПО. Десятки университетов в Северной Америке предлагают образование с получением степени в этой области и еще больше программ без получения степени.

Тот факт, что у нас не сложилось полностью устоявшихся знаний о практических методиках инженерии ПО, вряд ли уникален для этой сферы. В 30-х годах прошлого века медики еще не знали пенициллина, структуры ДНК и генетических основ многих болезней, не было аппаратов искусст-

венного дыхания и создания изображений посредством томографии. Однако медицинская профессия существовала.

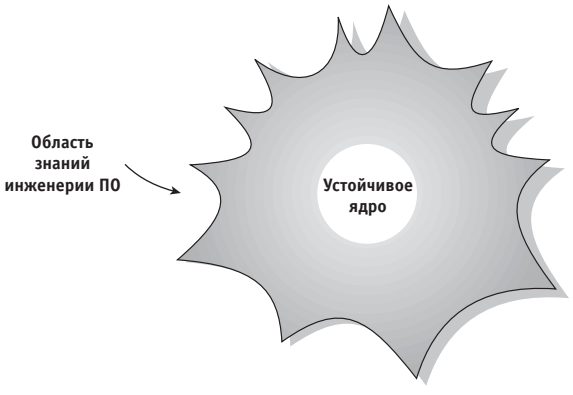

Рис. 5.1. *Со времен конференции НАТО по инженерии ПО в 1968 г. лишь от 10 до 20% области знаний инженерии ПО было устойчивым (т. е. оставалось бы существенным 30 лет спустя). Полупериод жизни области знаний инженерии ПО в тот момент составлял 10 лет*

Как показывает рис. 5.2, основанный на моем анализе области знаний проекта SWEBOK, начиная с 2003 г. устойчивое ядро составляет около 50% знаний, необходимых для разработки систем ПО. Может показаться, что это не слишком кардинальное изменение по сравнению с 10–20% в 1968 г., но оно заставляет предположить, что полупериод жизни области знаний вырос примерно с 10 до 30 лет. Это означает, что вложение средств в обра-

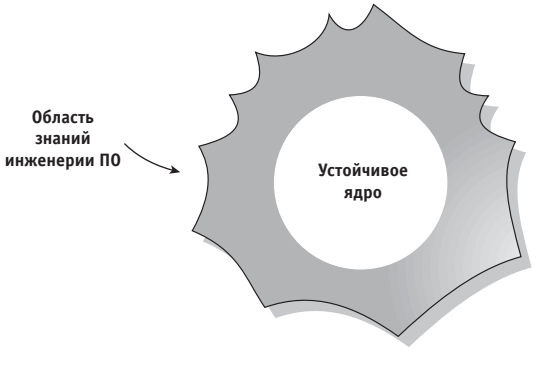

Рис. 5.2. *С 2003 г. около 50% области знаний инженерии ПО является устойчивым и останется существенным еще в течение 30 лет*

зование лиц, выбравших карьеру в сфере создания ПО, сделанное в самом начале их деятельности, останется в целом оправданным на протяжении всей карьеры.

Стабилизация области знаний инженерии ПО ставит эту сферу в равное положение с другими инженерными дисциплинами. Как отметил Дэвид Парнас, даже если в лаборатории физики появится новый осциллограф, содержание уроков физики при этом останется неизменным. Большинство учебных курсов инженерии ПО может не зависеть от конкретных технологий с относительно коротким сроком существования, таких как, например, C++ или Java. Студенты могут осваивать эти средства на практических занятиях, но теоретические занятия надо посвятить более долговечным знаниям.

Область знаний инженерии ПО

В главе 4 я утверждал, что инженерия ПО не то же самое, что компьютерная наука. Если это так, то что же такое инженерия ПО? Тем из нас, кто работает в сфере разработки ПО, предоставлена уникальная возможность наблюдать за рождением новой сферы деятельности. В уже сформировавшихся науках, таких как математика, физика или физиология, мы склонны принимать содержание науки как данность, считая, что оно всегда было таковым и таким же должно и оставаться. Но в какой-то момент авторам учебников и составителям университетских программ приходилось решать, что надо включить в курс (учебник), а что исключить. Веками люди не выделяли математику, физику, физиологию и философию как отдельные науки. Математика стала рассматриваться отдельно от философии примерно за 300 лет до нашей эры. Физика выделилась из философии где-то на рубеже XVI–XVII вв. Физиология не выделялась из философии примерно до 1900 г.

При определении того, что входит, а что не входит в предмет инженерии ПО, специалисты рекомендуют сосредоточиться на общепринятых знаниях и практических методах. Как следует из рис. 5.3, «общепринятое» относится к знаниям и практике, применимым к большинству проектов почти всегда, то есть к методикам, ценность и полезность которых признает большинство специалистов. Общепринятость не означает, что такие знания и практические методы универсально применимы во всех проектах. Лицо, ответственное за проект, по-прежнему определяет методики, наиболее приемлемые в данном случае [41].

Рис. 5.3. *Возможные категории в области знаний инженерии ПО. При определении знаний, нужных профессиональному инженеру ПО, общепринятые элементы должны быть в фокусе внимания. Источник: [134]*

С 1968 г. удалось значительно продвинуться в сферах, которые Брукс назвал «существенными трудностями» в разработке ПО. Сейчас имеются хорошие или хотя бы удовлетворительные справочники по спецификациям, конструированию, созданию, тестированию, экспертизе, обеспечению качества, управлению проектами ПО, алгоритмам и проектированию интерфейса пользователя – это лишь некоторые темы. Постоянно появляются новые более качественные издания, систематизирующие накопленные знания в этой сфере. Некоторые компоненты устойчивого ядра знаний пока не сведены в практические руководства или учебные курсы, и в этом смысле данная область знаний все еще формируется. Однако основная база знаний о применении описанных практических методик уже содержится в журнальных статьях, докладах на конференциях, трудах семинаров и в книгах. (Обширный перечень этих книг приводится на профессиональном сайте инженерии ПО по адресу *www.construx.com/profession*.) Пионеры инженерии ПО уже наметили пути и исследовали территорию. Сегодня прибывшие на «постоянное поселение» специалисты должны проложить дороги и развить недостающие образовательную и аттестационную инфраструктуры.

Исследователи Университета Квебека в Монреале возглавили работу по определению общепринятых элементов инженерии ПО. Их усилия координировали ассоциация ACM и IEEE Computer Society и привлекли как ученых, так и представителей отрасли. В целом эта работа получила назва-

ние «Guide to the Software Engineering Body of Knowledge» (Руководство к своду знаний по программной инженерии), или сокращенно SWEBOK.[1]

Как показано на рис. 5.4, инженерия ПО черпает знания из компьютерной науки, математики, наук о мышлении (психологии и социологии), управлении проектами и из различных инженерных дисциплин.

Рис. 5.4. *Источники информации в области знаний инженерии ПО. Источник: [134]*

В качестве стартовой позиции проект SWEBOK определил области знаний, позволяющие получить базовые умения и навыки профессионального инженера ПО [1].

- *Требования к ПО*. Выявление, описание и анализ требований, которым должно удовлетворять программное обеспечение.
- *Проектирование ПО*. Определение базовой структуры системы на архитектурном и детализированном уровнях, разбиение на модули, определение интерфейсов модулей и выбор алгоритмов внутри модулей.
- *Создание ПО*. Реализация ПО, включая детальное проектирование, написание программ, отладку, тестирование компонент, технические экспертизы и оптимизацию производительности. В какой-то степени этот пункт пересекается с проектированием и тестированием ПО.

[1] Дополнительная информация имеется на сайте проекта SWEBOK по адресу *www.swebok.org*.

- *Тестирование ПО.* Все действия, связанные с прогоном программ для оценки их возможностей и обнаружения дефектов. Тестирование включает его планирование, схему конкретных испытаний и специальные виды тестирования, включая тесты разработки, интегрирования, системные тесты, регрессионные тесты и приемные испытания.
- *Обслуживание ПО.* Обновление и расширение готового ПО, соответствующая документация и тестирование.
- *Управление конфигурацией ПО.* Определение, документирование и контроль изменений по версиям для всей интеллектуальной собственности, созданной в ходе проекта, включая исходный код, ресурсы (графика, звук, текст и видео), требования, проекты, материалы тестирования, оценки, планы и пользовательскую документацию.
- *Качество ПО.*[1] Все работы, связанные с обеспечением соответствия техническим требованиям компонент ПО. Управление качеством включает планирование мероприятий по его обеспечению и измерению, обеспечение надежности, тестирование, технические экспертизы, аудиты, а также сверку и утверждение.
- *Управление разработкой ПО.* Планирование, отслеживание и контроль проекта по разработке ПО, работ по созданию ПО или организации, занимающейся его разработкой.
- *Инструменты и методы разработки ПО.* Поддержка методологии и инструментария, таких как CASE-средства, прикладные библиотеки пользователя и формальные методики, включая практику внедрения и распространения методов и инструментария внутри организации.
- *Процесс разработки ПО.* Работы, связанные с повышением качества процессов разработки ПО, с соблюдением сроков, повышением производительности и других характеристик проекта и продукта.

Столь обширный список может кого-то удивить. Многие практики программирования работают так, как будто единственный значимый компонент – это создание ПО. Но каким бы важным ни был этот компонент, он всего лишь один из десяти, в которых следует разбираться профессиональному инженеру-разработчику ПО.

---

[1] Принятый в настоящее время русскоязычный термин «управление качеством» в гораздо меньшей степени акцентирован на инженерном подходе к этому процессу, чем в позиции автора. – *Примеч. науч. ред.*

Других практикующих программистов удивит полное отсутствие упоминания о конкретных языках и средах программирования – Java, C++, Visual Basics, Linux и т. д. Но дело в том, что область знаний концентрируется на принципах инженерии ПО, а не на знаниях технологий.

Кто-то, взглянув на этот список, скажет: «Сколько же надо знать, чтобы всего лишь писать компьютерные программы». Действительно, научиться надо многому, и всегда считалось, что знания будут осваиваться подспудно, при получении новой информации в процессе работы. В результате большинство программистов-практиков достаточно хорошо владеют созданием и обслуживанием ПО; отрывочными сведениями о требованиях, проектировании, тестировании и инструментах, методах разработки ПО, но практически ничего не знают об управлении конфигурацией ПО, качестве, управлении разработкой ПО и процессе разработки ПО.

Я не надеюсь на то, что инженеры ПО в совершенстве овладеют каждым из этих компонентов, но профессиональному инженеру ПО следует хотя бы ознакомиться с ними, разбираться в большинстве из них, а некоторыми овладеть в совершенстве.

Как сказано в главе 4, одно из отличий ученого от инженера состоит в том, что ученый может позволить себе владеть узким, но глубоким знанием, а инженер должен знать и учитывать все факторы, которые воздействуют на создаваемый им продукт.

Ставим зарубку

Является ли наше определение свода знаний инженерии ПО окончательным? Нет. Инженерия ПО продолжает развиваться, как и медицина и другие науки. Но очень важно поставить зарубку и сказать: «Вот это и есть свод знаний инженерии ПО на данный момент».

Как заметил Френсис Бэкон, когда 350 лет назад он закладывал основы современной науки, ошибки создают более прочную основу для развития, чем путаница. Он осознавал, что его подход сделает многие первоначальные выводы ошибочными, но это было частью его плана. Главные элементы сегодняшнего определения свода знаний инженерии ПО несомненно окажутся ошибочными, однако оно даст точку отсчета для дальнейшего совершенствования. Замена существующей путаницы и неразберихи на четко сформулированное понятие области знаний – это удачный обмен, несмотря на возможные ошибки и прочее.

ГЛАВА ШЕСТАЯ

Новый органон

Умный вопрос – это уже добрая половина знания.
Френсис Бэкон

В 1620 г. Френсис Бэкон опубликовал свой эпохальный труд «Новый органон» (Novum Organum), в котором бросил вызов своим современникам, предложив им отказаться от привычки полностью полагаться на дедуктивное мышление и принять научный метод исследования, основанный на наблюдении и опыте. В его воображении рождался новый мир, появление которого он связывал с проникновением в законы и процессы природы. Описывая этот мир, Бэкон предвидел результаты достижений в науке, инженерии и технологии.

Следование научному методу Бэкона предполагало три шага:

1. Отказаться от предубеждений («предрассудков»).
2. Систематически накапливать наблюдения и опыт.
3. Остановиться, обозреть видимое и сделать первоначальные выводы – «первый урожай».

«Новый органон» был частью монументального труда, названного «Великое восстановление наук» (Instauratio Magna), в котором делалась попытка упорядочить науки, сформулировать метод научного познания, накопить наблюдения и факты, дать примеры новых методов и создать новую философию, основанную на результатах такого научного подхода. Этот труд оказал столь фундаментальное влияние на современные научные методы, что Бэкона часто называют отцом современной науки.

На заглавной странице этого труда был изображен корабль между Геркулесовыми столпами (рис. 6.1), которые, по преданию, стояли по обоим берегам Гибралтарского пролива – единственного пути из Средиземного моря в Атлантику. В древние времена эти столпы были символом земного предела люди: боялись выйти за этот предел и оказаться во внешнем пространстве, оставив свой мир.

Рис. 6.1. *Фронтиспис труда Френсиса Бэкона «Великое восстановление», включающего «Новый Органон». Античные мореплаватели не стремились выходить за пределы, ограниченные Геркулесовыми Столпами*

Принятые сейчас методы разработки ПО подобны кораблю, застрявшему в безветренном море – море программирования, где господствует принцип «написать и исправить». Подобный подход сродни представлению о плоской Земле, показавшему свою неэффективность еще 20 лет назад. Лидеры разработки ПО хорошо осознавали, что лежит за Геркулесовыми столпами – инженерия ПО, где уже появились свои Марко Поло, Васко да Гама и Фернан Магеллан, исследовавшие новый мир совершенных методик разработки ПО. Как описано в главе 12, огромные богатства мето-

дов написания ПО лежат в морях, которые хорошо изучены, но куда редко заходят корабли обычных практик ПО.

Формирование профессии

С учетом огромного разрыва между лучшими и худшими фирмами, занимающимися разработкой ПО, задача сегодняшнего дня заключается не столько в том, чтобы развить передовые методики, сколько в том, чтобы подтянуть к ним средний уровень разработки ПО. Новый мир достаточно хорошо изучен, пора уже его освоить. Традиционный способ повышения среднего уровня в отрасли, напрямую затрагивающей жизнедеятельность общества, состоит в создании официальной профессии.

Хотя само слово «профессия» часто употребляется небрежно, стоящее за ним понятие имеет давно устоявшееся юридическое значение. В «Своде федеральных постановлений США» (СФП) говорится, что лицо, выполняющее «профессиональные обязанности», отличают несколько характеристик.

Как правило, профессиональная работа требует глубоких знаний науки или обученности в какой-либо области, которые приобретаются в процессе продолжительной специальной учебы и подготовки. В СФП делается различие между профессиональной подготовкой и общим научным образованием, а также подготовкой к работе согласно стандартным процедурам, будь то мыслительный, ручной или физический труд. Далее в СФП указывается, что профессиональная работа может быть творческой и художественно-изобразительной. Это зависит главным образом от изобретательности, воображения или таланта исполнителя.

По определению СФП профессиональная работа требует последовательной реализации собственных суждений и права выбора. Ей свойственен интеллектуальный характер и смена форм. И снова, как мы видим, СФП отличает ее от рутинного мыслительного, ручного или физического труда.

Многие разработчики ПО увидят в определении профессии, даваемой в СФП, характеристики своей собственной работы. Эта работа, несомненно, требует глубоких знаний (или в любом случае подробных технических сведений), и ее возможности связаны со специальным обучением и подготовкой. Разработка ПО содержит значительный элемент творчества и, конечно, требует применения суждений и права выбора. Таким образом, работа, выполняемая разработчиками ПО, отвечает определению «профессиональной работы», данному в Своде федеральных положений США.

Но СФП лишь формулирует определение профессии. Судебные прецеденты (реальные судебные дела) наполняют его несколько иным содержанием. Согласно им, профессия – это [69]:

- Глубокие познания и серьезная подготовка.
- Кодекс поведения, устанавливающий более высокие этические стандарты, чем те, которые обычно допускаются на рынке.
- Система дисциплинарных наказаний для профессионалов, нарушающих этот кодекс.
- Высокая социальная ответственность по сравнению с личной выгодой и соответствующая обязанность соблюдать профессиональную этику.
- Необходимость получения лицензии перед началом практической деятельности.

Так насколько же соответствует этим критериям разработка ПО?

В поисках профессии инженерии ПО

Гэри Форд (Gary Ford) и Норман Е. Гиббс (Norman E. Gibbs) из Института инженерии ПО сформулировали восемь элементов зрелой сформировавшейся профессии [46]:

Начальное профессиональное образование. Профессиональные работники обычно начинают профессиональную карьеру с прохождения университетской программы в избранной ими области – в медицинском, инженерном, юридическом вузе и т. д.

Аккредитация (аттестация). Аттестация университетских программ осуществляется органами надзора, которые определяют степень соответствия программ целям образования. Этим обеспечивается наличие необходимой подготовки у выпускников вузов, дающей возможность начать профессиональную деятельность по его окончании. Совет по аккредитации программ в области техники и технологии (Accreditaton Board for Engineering and Technology – ABET) осуществляет надзор за программами технических дисциплин в США. Аналогичная организация – Инженерный аккредитационный совет (Canadian Engineering Accreditation Board – CEAB) – действует в Канаде.

Развитие практических навыков и приемов. В большинстве профессий для овладения профессиональными навыками одного только образования недостаточно. Начинающим профессионалам необходима практи-

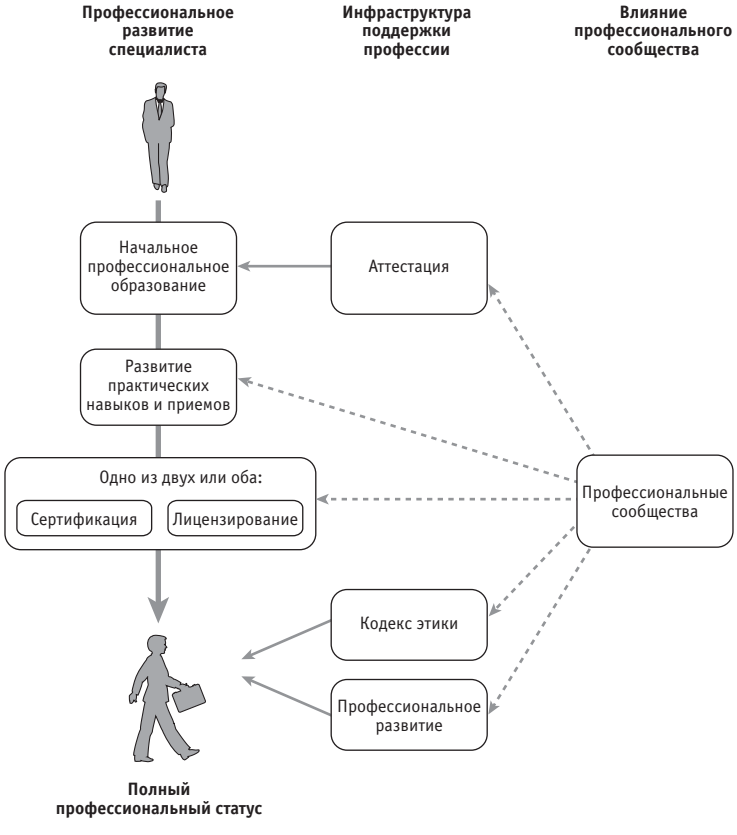

Рис. 6.2. *В сформировавшихся отраслях профессиональное развитие проходит через все основные ступени (или их большинство)*

ка применения своих знаний, перед тем как принять на себя ответственность за выполнение работ в выбранной области. У врачей предусмотрено три года ординатуры. Бухгалтеры (CPA) должны один год отработать в утвержденной специальным комитетом организации, чтобы получить лицензию. Профессиональные инженеры должны иметь по крайней мере 4 года опыта практической работы. Обязательный период ученичества гарантирует, что в профессию попадут люди, имеющие опыт выполнения работы на требуемом уровне компетентности.

Сертификация. Завершив образование и развив практические навыки, профессионал должен сдать один или несколько экзаменов, чтобы доказать, что он овладел необходимым минимумом знаний. Врачи сдают

экзамены в соответствующих профессиональных советах. Бухгалтеры сдают экзамен на звание (Certified Public Accountant – CPA). Инженеры сначала сдают экзамен по основам инженерии по окончании колледжа, а затем, примерно через четыре года, экзамен по специальности. В некоторых профессиях требуется подтверждать сертификат через определенные периоды времени.

Лицензирование. Лицензирование аналогично сертификации, за тем исключением, что оно носит обязательный характер и осуществляется государственными органами.

Профессиональное развитие. От профессионалов требуется актуализировать свое специальное образование. Непрерывное профессиональное образование поддерживает на необходимом уровне или улучшает знания и умения работников, после того как они приступили к профессиональной деятельности. Требование профессионального развития особенно сильно в тех специальностях, где область технических знаний подвержена быстрым переменам. Здесь пальму первенства удерживает медицина – в силу постоянного совершенствования лекарственных препаратов, терапии, медицинского оборудования, методов диагностики и лечения. Требование профессионального развития помогает обеспечить сохранение минимально необходимого уровня компетенции на протяжении всей карьеры.

Профессиональные сообщества. Профессионалы считают себя членами сообществ, состоящих из подобных им индивидуумов, которые ставят профессиональные стандарты выше своих личных интересов или интересов работодателей. Поначалу профессиональные сообщества содействуют обмену опытом и знаниями, но со временем их функции расширяются и включают определение критериев сертификации, управление ее программами, формулировку стандартов аттестации и этических кодексов, а также мер дисциплинарного воздействия на нарушителей этих кодексов.

Кодексы поведения. В каждой профессии существует этический кодекс, регулирующий поведение людей, принадлежащих к данному сообществу профессионалов. В кодексе не просто указывается, что должно делать, а что нет. За его нарушение профессионалы могут быть исключены из своих профессиональных сообществ или лишены лицензии на профессиональную деятельность. Соблюдение признанного этического кодекса поведения помогает профессионалам ощущать свою

принадлежность к уважаемому сообществу, а меры по обеспечению этических стандартов помогают поддерживать некий минимальный приемлемый уровень поведения.

Помимо восьми характерных элементов профессии, выделенных Фордом и Гиббсом, во многих профессиях появляется девятый, который скорее относится к целым организациям, нежели к отдельным работникам:

Сертификация организаций. Во многих отраслях лицензию должны получать не только отдельные профессионалы, их организации тоже должны быть сертифицированы. Бухгалтерские фирмы проходят проверки такими же организациями. Больницы и университеты тоже аттестуются. В таких сложных сферах деятельности, как бухгалтерский учет, образование и медицина, сертификация организаций гарантирует надлежащий уровень выполнения работы, который не может быть достигнут за счет компетентности только индивидуумов. Качества, характеризующие саму организацию, могут влиять на этот уровень так же сильно, как и характеристики отдельных работников.

Форд и Гиббс указывают, что многие непрофессиональные виды деятельности обладают определенным количеством перечисленных элементов-характеристик. Например, в штате Калифорния требуется иметь лицензии обивщикам, боксерам-любителям, частным детективам и жокеям в скачках на мулах, но эти виды деятельности не требуют большинства других элементов профессиональной работы. Все обычные профессии обладают почти всеми названными элементами.

Для каждого из элементов, формирующих профессию, Форд и Гиббс также определили несколько уровней зрелости:

Отсутствие. Данный элемент просто не существует.

По необходимости. Элемент существует, но только в отдельных невзаимосвязанных случаях.

Сформировавшийся. Элемент существует и четко определяется в конкретной (specific) профессии. (Форд и Гиббс употребляют термин «конкретный» , но мне кажется, что он лишь сбивает с толку.)

Зрелый. Элемент существует в течение длительного времени и активно поддерживается и совершенствуется каким-либо профессиональным органом.

Зрелость профессии означает, что ее элементы достигли необходимой стадии. Разумеется, понятие зрелости постоянно меняется. Некоторые эле-

менты, казавшиеся зрелыми лет 30 назад, сегодня уже не представляются таковыми, а нынешняя зрелость других не будет признаваться через 30 лет.

В табл. 6.1 описывается зрелость профессии инженерии ПО. В основном профессия сформировалась, но некоторые элементы поотстали, а некоторые продвинулись к стадии зрелости. Положения, сформулированные в таблице, обсуждаются более подробно далее.

Таблица 6.1. Зрелость элементов профессии инженерии ПО [46]

Элемент	Текущее состояние
Начальное профессиональное образование	В промежутке между «по необходимости» и «сформировавшимся». Степень бакалавра компьютерной науки, электротехники, математики и т. п. составляет обычную подготовку для вступления в профессию. Имеются десятки программ для получения степени магистра инженерии ПО. В последние годы инициированы многочисленные программы для студентов старших курсов, но пока лишь немногие успели завершить эти программы.
Аккредитация (Аттестация)	Сформировавшийся. Указания по аттестации сейчас формулируются рабочей группой от IEEE Computer Society и ACM, но еще не реализованы.[a]
Развитие практических навыков и умений	Сформировавшийся. Разработаны методические указания по навыкам и умениям, которыми должен обладать инженер ПО, чтобы начать заниматься этой профессией.
Сертификация	Сформировавшийся. Коммерческие поставщики ПО, такие как Microsoft, Novell и Oracle, уже многие годы предлагают программы сертификации, связанные с технологиями. С 2002 г. IEEE Computer Society предлагает Свидетельство сертифицированного профессионального разработчика ПО как подтверждение общего статуса в инженерии ПО.
Лицензирование	По необходимости. В штате Техас профессиональные разработчики ПО лицензируются согласно положению, принятому в 1998 г. В провинциях Британская Колумбия и Онтарио регистрация профессиональных инженеров ПО началась в 1999 г.

Элемент	Текущее состояние
Профессиональное развитие	По необходимости; тенденция приближения к сформировавшемуся. Некоторые организации опубликовали методические указания по профессиональной разработке ПО. (См. основные направления непрерывного образования IEEE Computer Society на сайте *www.computer.org/certification* и такой же документ фирмы Construx на сайте *www.construx.com/ladder*.)
Профессиональные сообщества	Сформировавшийся уровень с тенденцией приближения к зрелости. Существуют IEEE Computer Society, ACM (Association for Computer Machinery) и другие профессиональные ассоциации. Эти сообщества прямо указывают, что они представляют инженерию ПО. Весь комплекс продуктов и услуг, необходимых для поддержки инженеров ПО как профессионалов, эти сообщества не предлагают. Они не могут принимать дисциплинарные меры к нарушителям этического кодекса инженерии ПО.
Кодекс этического поведения	Сформировавшийся. ACM и IEEE Computer Society приняли кодекс этического поведения для инженеров ПО. Этот документ пока не принят в отрасли и не получил широкого признания.
Сертификация организаций	Сформировавшийся уровень с тенденцией приближения к зрелости. Институт инженерии ПО сформулировал модель зрелости в разработке ПО, которую он активно продвигает и совершенствует. С 1987 г. модель использовалась для оценки более полутора тысяч организаций, но повсеместно пока не применяется.[b] Сертификация согласно ISO 9000-9004 широко признана, особенно в Европе.

[a] Текущий статус описан на сайтах *www.computer.org/ccse* и *www.computer.org/education/*

[b] Следует отметить, что за последние два года популярность сертификации по моделям качества процессов разработки ПО CMM и CMMI, разработанным Институтом инженерии ПО, резко возросла. Особенно это относится к индийским, ирландским и российским компаниям-разработчикам ПО, которые связывают сертификацию на 4 и 5 уровнях модели качества процессов CMM и CMMI с укреплением своих позиций на рынке «офшорного программирования». – *Примеч. науч. ред.*

Проход через Геркулесовы столпы

Инженерия ПО пока не полностью отвечает определению профессии. Еще только формируется система доступного начального образования. Сертификация получила распространение лишь в 2002 г. Лицензирование доступно лишь крошечной доле занятых в разработке ПО. Этический кодекс существует, но соблюдение его положений не обеспечивается применением дисциплинарных мер. Ведется большая работа по ускорению продвижения инженерии ПО к сформировавшейся и зрелой стадиям.

Применив научный метод Бэкона к инженерии ПО, можно определить три шага, которые необходимо сделать, чтобы помочь инженерии выйти на уровень зрелости:

Освободить мышление от предрассудков. Отрасль ПО должна отказаться от пагубного пристрастия к разработке ПО по принципу «написать и исправить», о котором давно известно, что он никому не выгоден.

Систематически накапливать наблюдения и опыт. В фирмах начали систематизировать данные об эффективности практических методик разработки ПО и оценивать, какие из них приносят наибольший успех. Причем некоторые фирмы достигли впечатляющих результатов. Другие должны последовать их примеру.

Остановиться, посмотреть, что получено, и сделать первоначальные выводы. Некоторые из таких выводов сделаны в данной книге.

Сфера разработки ПО подошла к одному из поворотных пунктов. Можно остаться в удобном мирке программирования по принципу «напишем и исправим», не пытаясь вырваться за пределы Геркулесовых столпов и не стараясь добиться огромных преимуществ, которые сулят исследования лидеров ПО. Но можно смело рвануться вперед к новой профессии инженерии ПО и начать обживаться в мире высокой производительности, снижения расходов, укороченных сроков разработки и более высокого качества.

ЧАСТЬ ВТОРАЯ

Индивидуальный профессионализм

ГЛАВА СЕДЬМАЯ

«Предпочтение отдается сиротам»

Требуются молодые, жилистые, худые парни не старше 18 лет. Обязательно опытные наездники, готовые ежедневно рисковать жизнью. Предпочтение отдается сиротам. Зарплата 25 долларов в неделю.

Объявление о приеме на работу компании курьерских и грузовых перевозок PONY EXPRESS, 1860 г.

Мы отдаем себе отчет, что квалификационные, интеллектуальные и личностные качества нужных нам кандидатов чрезвычайно редки, и это отражается в предлагаемых вознаграждениях. В свою очередь,мы рассчитываем на ПОЛНОЕ И АБСОЛЮТНОЕ ПОДЧИНЕНИЕ успеху проекта – преодоление всех препятствий ради создания программного продукта в установленный срок и в рамках выделенных средств.

Объявление о приеме на работу разработчиков ПО, 1995[1]

Сложившийся стереотип представления о программисте – это застенчивый юноша, работающий в полутемной комнате, глубоко сосредоточившись на магических формулах, которые заставляют компьютер выполнять его волю. Программист может работать без отрыва по 12–16 часов, часто просиживает ночами, реализуя свои художественные замыслы. Питается пиццей и «Твиксами». Если его оторвать от работы, начинает бурно выражать эмоции, выкрикивает, адресуясь к источнику беспокойства, загадочные аббревиатуры, вроде TCP/IP, RPC, RCS, ACM и IEEE. Программист может

[1] *The Seattle Times*, October 8, 1995. Выделено в оригинале.

нарушить свою сосредоточенность лишь для участия в конгрессах любителей Star Trek или чтобы посмотреть повторный прогон «игрушки» Monty Python. Иногда его считают незаменимым гением, иногда эксцентричным художником. В его и только в его голове хранится вся необходимая информация. Он спокоен за свою работу, хорошо понимая, что в силу его нужности лишь горстка достойнейших может претендовать на его место.

Издание *USA Today* сообщило, что распространенный стереотип фаната-компьютерщика настолько укоренился в сознании людей, что студенты всех курсов называли эту специальность среди последних при выборе карьеры.[1] Газета *Wall Street Journal* указывала на трудности, с которыми сталкивались съемочные группы, пытаясь сделать интересный материал о ведущих компаниях ПО, поэтому каждый раз снималась картинка – офисный центр, кабинка, стол с каким-то ящиком и сидящий за ним сотрудник [18]. Иногда этот стереотип подпитывается и самими профессионалами. Газета *The New York Times* цитировала слова, якобы сказанные содиректором программы «компьютероведения» Стэнфордского университета, что занятие программным обеспечением «нагоняет скуку, убивающую интеллект».[2] И все это несмотря на то, что профессии, связанные с ПО, получают высшие рейтинги в таких авторитетных источниках, как ежегодное издание *Jobs Rated Almanac* [73].

Насколько верен этот стереотип и как он влияет на профессию программиста? Взглянем сначала на личность программиста, а затем на другие составляющие этого стереотипа.

Характеристики типа личности по Майерс-Бриггс

Распространенный способ классификации личности разработали Кэтрин Бриггс (Katherine Briggs) и Изабель Майерс-Бриггс (Isabel Briggs Meyers)). Способ так и называется – классификатор типа личности по Майерс-Бриггс, или сокращенно MBTI. Согласно этому методу, люди делятся на типы личности по четырем классификационным признакам:

Экстраверсия (Extraversion – E) или интроверсия (Introversion – I). Первые сосредоточены на окружающем мире людей и вещей. Вторые больше погружены в мир воображаемых идей.

---

[1] *USA Today*, February 16, 1998, pp. 1B–2B.

[2] «Software Jobs Go Begging», *The New York Times*, January 13 1998, p. A1.

Ощущения (sensing – S) или интуиция (Intuition – N). Этот признак относится к предпочитаемому способу получения информации для принятия решений. Человек, полагающийся на ощущения, сосредотачивается на известных фактах, конкретных данных и опыте. Интуитивная личность ищет возможности в представлениях и теориях.

Мышление (Thinking – T) или чувства (Feeling – F). Этот признак относится к методу принятия решений. В первом случае решения принимаются на основе объективного анализа и логики, во втором – на основе субъективных чувств и эмоций.

Восприятие (Perceiving – P) или суждение (Judging – J). Воспринимающая личность предпочитает гибкость и открытые возможности, тогда как человек, выбирающий суждения, стремится к контролю и порядку.

Чтобы определить тип личности по методу Майерс-Бриггс (MBTI), надо выполнить тест, в процессе которого присваивается один из четырех признаков категории, то есть общий тип обозначается четырьмя буквами, например ISTJ или ENTJ. Эти буквы указывают на наклонности и (или) тенденции поведения личности. При этом они не определяют поведение человека в конкретной ситуации. Несмотря на наличие черт интроверта, можно развивать характеристики экстраверта, чтобы чувствовать себя более свободно, например, в деловом окружении. Такой человек получит признак «*I*», хотя большинство коллег назовут его экстравертом.

Результаты теста MBTI разработчиков ПО

По результатам двух крупных исследований определился наиболее характерный для разработчиков ПО тип личности – ISTJ («интроверт, опирающийся на ощущения, мыслитель, склонный к суждениям») [81], [133]. Этот тип личности склонен к серьезности и спокойствию, практичен, аккуратен, логичен и достигает успеха за счет сосредоточенности и скрупулезности. От 25 до 40% разработчиков ПО принадлежат именно к этому типу [16], [81], [133].

В соответствии со сложившимся стереотипом программисты – действительно интроверты. От одной до двух третей сообщества разработчиков ПО – интроверты по сравнению с примерно одной четвертой долей интровертов среди людей в целом [81], [133]. Частично этот факт может объясняться большей долей интровертов среди стремящихся получить высшее образование, а программисты в большинстве своем более образован-

ны, чем население в среднем. Около 60% разработчиков ПО имеют степень бакалавра по сравнению с 30% прочих людей.[1]

Признаки S/N (ощущения/интуиция) и T/F (мышление/чувства) особенно интересны, поскольку характеризуют личностный стиль принятия решений. От 80 до 90% разработчиков ПО относятся к типу T по сравнению с 50% среди прочего населения [16], [81], [133]. По сравнению со средним уровнем «мыслители» более логичны, аналитичны, им свойственен научный поход, они бесстрастны, хладнокровны, беспристрастны, их больше волнует правда, чем чувства людей.

Что касается S и N, то здесь разработчики ПО делятся почти поровну, и разница между двумя типами сразу же распознается ими же самими. Люди типа S живут в фактическом мире достижимого сегодня, они точны, конкретны и практичны, стремятся к специализации, склонны глубоко разрабатывать одну идею, а не несколько замыслов сразу. Люди типа N живут в мире возможного, теоретического, склонны к обобщениям и исследованию нескольких противоположных идей. Примером S-типа может служить прекрасный программист, в деталях освоивший конкретный язык программирования или технологию. Пример N-типа – разработчик, рассматривающий широкий спектр возможностей и отмахивающийся от технических деталей как от несущественных проблем реализации. Наличие личностей S-типа часто подчеркивается присутствием людей N-типа, поскольку первые вникают в технические детали раньше, чем вторые почувствуют, что вопрос достаточно исследован вширь. В обратной ситуации первые (N-тип) отсеивают одно решение за другим раньше, чем вторые (S-тип) смогли достаточно глубоко изучить конкретную техническую область.

Личные качества великих изобретателей

Показатели MBTI позволяют глубже рассмотреть типичные личностные качества программистов, однако по ним нельзя сделать окончательный вывод. Важной группой качеств разработчика ПО являются навыки инженерии ПО. Многие программисты стремятся стать выдающимися разработчиками. В чем же главные черты великих изобретателей? Согласно исследованию, посвященному великим изобретателям в целом [52] (не только создателям ПО), наиболее творчески мыслящие представители

[1] National Center for Education Statistics, *2001 Digest of Educational Statistics*, Document Number NCES 2002130, April 2002.

этой категории, дающие ответы на проблемы человечества, с кажущейся легкостью дрейфуют в области между крайними точками S/N, T/F и P/J. Они могут уходить от всестороннего подхода к узко направленному, от интуитивного мышления к логическому, от теории к деталям. Примерами таких великих творцов были Леонардо да Винчи и Альберт Эйнштейн (хотя вряд ли им приходилось выполнять тест MBTI).

Великие изобретатели владеют массой стандартных приемов для решения каждой новой задачи. Если проблема вписывается в готовую методику, она может быть легко решена уже известным способом.

Великие изобретатели в совершенстве владеют своими инструментами.

Они не боятся сложностей, наоборот, великие сложности их только притягивают. Однако их цель состоит в том, чтобы упростить то, что кажется сложным. Великий Эйнштейн говорил, что все должно быть простым, насколько это возможно, но не проще. Французский писатель и летчик Антуан де Сент-Экзюпери утверждал то же самое, сказав, что совершенство достигается не тогда, когда больше нечего добавить, а когда больше ничего нельзя отнять.

Великие изобретатели ищут оппонентов для своих работ. Возникающая таким образом обратная связь позволяет испробовать и отвергнуть многие возможные решения.

У великих изобретателей, конечно, бывают и неудачи, но они стараются учиться на ошибках. Испытываются и отбрасываются различные варианты, часто совершаются ошибки, но великие изобретатели находят свои недочеты и исправляют их. Они упорно продолжают придумывать новые варианты, даже когда другие давно уже оставили попытки решить проблему.

Великие не избегают применения грубой силы для решения задачи. Томас Эдисон искал материал для нити лампочки накаливания почти два года, испытав тысячи различных материалов. Как-то помощник спросил его, как у него хватает сил продолжать поиск после стольких неудач. Эдисон просто не понял вопроса. В его понимании неудач не было. Он мог бы ответить: «Какие неудачи? Я теперь знаю тысячи материалов, которые не годятся».

Великие изобретатели должны обладать творческим мышлением, позволяющим выдвигать множество вариантов решения проблемы. Основой массы исследований является творчество, и здесь прослеживаются некоторые общие черты. Великие изобретатели любопытны, но их любопытство носит расширенный характер. Они энергичны, достаточно уверены в себе и настолько независимы в своих суждениях, что изучают и исполь-

зуют концепции, которые считаются глупостью. Их интеллектуальная честность позволяет им отделить свои собственные представления от того, что, по общепринятому мнению, им следовало бы думать.

У великих изобретателей есть неуемная тяга к *созиданию*. Это может быть строительство зданий, создание электронных схем или компьютерных программ. Они испытывают тягу к действию: простое познание реальности их не удовлетворяет, и они чувствуют обязанность применить свои познания к реальным ситуациям. Для великих изобретателей неприменение знаний равносильно незнанию.

Программисты живут ради открытий, приводящих к революционным решениям в проектировании ПО. Мне кажется, это одна из причин, делающих сходство разработчиков ПО и Монти Питона более обоснованным, чем это могло бы показаться на первый взгляд. Монти Питон насмехается над социальными условностями, опираясь на неортодоксальное противопоставление элементам современности и культуры. Точно такое же независимое креативное мышление, которое стимулирует создание сценариев у Монти Питона, рождает инновационные технические решения и в проектировании ПО, к которым стремятся программисты.

Некоторые из названных качеств вписываются в стереотипный образ программиста, другие нет. Незнакомым с разработкой ПО людям программирование может показаться сухим и нетворческим делом. Те же, кто знают программирование изнутри, отлично понимают, что самые громкие современные проекты были бы невозможны без высшей степени творчества в создании ПО. Анимация, исследования космоса, компьютерные игры, медицинские технологии – трудно назвать передовые сферы деятельности человека, которые не опирались бы на творчество разработчиков ПО.

Но оставим стереотипы в стороне. Самим разработчикам ПО хорошо известно, что программирование дает возможность создавать нечто из ничего и приносит такое же удовлетворение, как создание скульптур или картин, написание книг или любая другая деятельность, творческий характер которой более очевиден. Так, говорите «нагоняет скуку, убивающую интеллект»? Не думаю.

Полная и абсолютная отдача

Стереотип представления о программисте, работающем по 12–16 часов в день, содержит больше чем зерно истины. Для того чтобы разработка ПО была эффективной, необходима способность сконцентрироваться ис-

ключительно на задаче программирования. Это требует жертв. Сконцентрировавшись на программировании, люди теряют ощущение времени. Оторвавшись от утренней работы, можно вдруг увидеть, что уже 2 часа – завтрак пропал. Посмотришь в пятницу на часы, а они показывают 11 часов вечера. Свидание не состоялось или вы забыли сказать жене, что придете поздно. Октябрьским днем до вас доходит, что лето закончилось, и вы опять пропустили его, потому что три месяца работали над интересным проектом.

Рекламное объявление курьерской фирмы в первом эпиграфе к этой главе могло бы быть смело применено сегодня к некоторым разработчикам ПО. Когда приходится работать так много, то под угрозу могут попасть семья, друзья или другие социальные привязанности. Вот как описывает самоотдачу программистов, работавших над созданием ОС Windows NT, Паскаль Захарий (Pascal Zachary) [146]:

> «Работа пронизала само их существование. Друзья отошли на второй план. Супружеские узы поистрепались или совсем разорваны. Детям не уделялось внимание или они отодвигались в сторону. Хобби забыты. Компьютерная программа стала значить все. Если и лелеялись какие-то личные мечты, то лишь как средство отвлечься от создания NT.»

После завершения проекта Windows NT одни разработчики ушли из компании, а другие были настолько опустошены, что вовсе ушли из программирования.

Осознавая эту опасность, некоторые опытные разработчики не всегда охотно берутся за отдельные проекты, отдавая себе отчет в том, что это означает потерянные вечера, выходные и пропавшее лето.

Однако всех этих жертв можно избежать, взяв на вооружение практику инженерии ПО. В среднем проекте от 40 до 80% времени уходит на исправление дефектов [13], [44], [63], [91], [140]. Практика инженерии ПО позволяет разработчикам прежде всего не делать ошибок, а сделанные устранять проще и быстрее. Уменьшить выполняемую работу наполовину – это очень простой способ сократить рабочую неделю с 80 до 40 часов.

Существует парадоксальная связь между отношением разработчиков к проекту и обязательствами перед компанией. По моему опыту, даже если программистам не нравится компания, в которой они работают, они редко уходят в середине проекта. Занятые в других сферах могут сказать: «Я ненавижу свою фирму. Дождусь середины проекта и уйду. Я им покажу.» Разработчики ПО мыслят по-другому: «Я ненавижу свою фирму. Закончим проект, и я уйду. Пусть видят, кого они теряют. Я им покажу.»

Программисты, по-видимому, еще и чрезвычайно преданы своему занятию. Одна из трудностей, с которыми сталкиваются компании при реализации договора о неразглашении информации, состоит в том, что многие программисты острее чувствуют общность со своими коллегами в других компаниях, чем испытывают лояльность к своим работодателям. Мне приходилось наблюдать, как многие разработчики ПО обсуждают конфиденциальные материалы компании с коллегами, не связанными договором о неразглашении. По их разумению, свободный обмен информацией между компаниями важнее, чем защита коммерческих секретов какой-то отдельной фирмы. Программисты, приверженные движению за открытый исходный код, воспринимают эту идею шире и отстаивают мнение, что все исходные коды и связанные с ними материалы должны раскрываться ради общественного блага [115], [128]. Мне кажется, что самоотречение ради проекта, продолжительная работа и высокие требования к креативности взаимосвязаны. Как только программист мысленно увидел программу, которую нужно создать, ее воплощение в жизнь становится доминантой, и программист не сможет успокоиться, пока не достигнет цели.

Способность полностью отдавать себя воплощению замысла свидетельствует о хороших перспективах формирования профессии инженерии ПО. Программисты ощущают стремление посвятить себя не только своим интересам, но и своим коллегам по проекту или даже всей отрасли ПО. Профессия инженерии ПО и связанные с ней профессиональные сообщества могут обеспечить этой преданности профессии созидательную направленность.

Демография ПО

Всложившемся стереотипе программиста как молодого мужчины есть определенная доля правды. Средний работник отрасли ПО моложе, чем в других отраслях в США. Как показано на рис. 7.1, кривая возрастной структуры работников отрасли ПО достигает пика в районе 30–35 лет, что примерно на 10 лет раньше, чем в других технических отраслях.

Большинство разработчиков ПО – мужчины. Последние данные (за 2000 г.) показывают, что 72% получивших степень бакалавра и 83% удостоенных степени кандидата наук в сфере компьютерной науки и информатики – это мужчины.[1] Только 17% выпускников школ, сдающих экзамен на

[1] National Center for Education Statistics, *2001 Digest of Educational Statistics*, Document Number NCES 2002130, April 2002.

продолжение обучения по компьютерной специальности, – девушки, а это низший показатель из всех специальностей.[1]

Итак, в среднем программисты моложе, чем специалисты других профессий, и в основном это мужчины. Сравнение с курьерами из Pony Riders выглядит все менее натянутым (хотя нет доказательств, что программисты в среднем более «жилистые»).

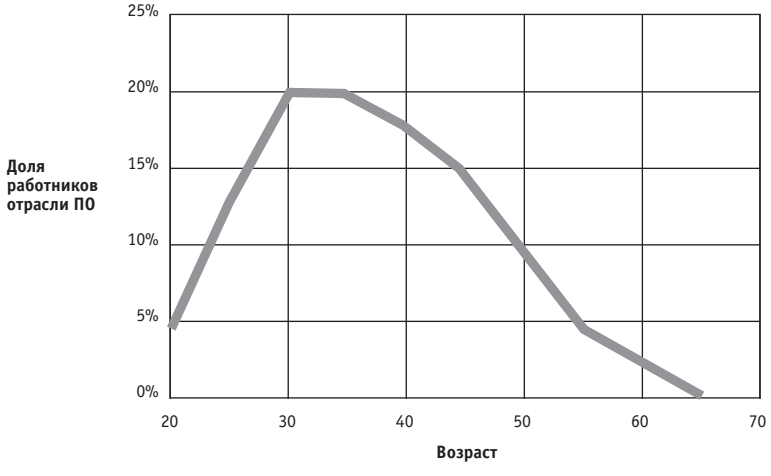

Рис. 7.1. *Возраст основной доли работников отрасли ПО от 30 до 35 лет, что на 10 лет меньше, чем в основных группах работников других технических отраслей. Источник: [80]*

Образование

Большинство программистов постепенно приобретает вкус к профессии. Когда я впервые написал небольшую программу, то думал, что как только программа оттранслируется и уйдут все синтаксические ошибки, я овладею программированием. Затем я перестал делать синтаксические ошибки, но иногда мои программы не работали так, как было задумано. Появлявшиеся проблемы оказались более сложными, чем ошибки в синтаксисе. Поэтому я стал думать, что как только смогу отлаживать программы, то овладею программированием. Так я думал, пока я не начал создавать достаточно большие программы, и отдельные участки и блоки не работали

[1] «Software Jobs Go Begging», *The New York Times*, January 13 1998, p. A1.

вместе так, как предполагалось. Тогда я стал думать, что стоит мне правильно разработать программу, как я наконец-то овладею искусством программирования. Я спроектировал несколько прекрасных программ, но приходилось менять их структуры и конструкции, потому что менялись требования. И я решил, что если смогу формулировать правильные требования, то тут уж точно стану настоящим мастером программирования. Но на пути к постижению этой науки я стал осознавать, что, возможно, никогда не овладею искусством разработки ПО. Это понимание стало первым реальным шагом к осознанию важности инженерии ПО.

Программисты разными путями приходят к осознанию этого факта. Некоторые идут по пути, похожему на мой, некоторые – другими дорогами. Большинство разработчиков хорошо образованы в целом, но есть и самоучки. Как показывает табл. 7.1, около 60% разработчиков ПО имеют степень бакалавра или более высокую степень. По данным фонда United Engineering Foundation, около 40% занятых в отрасли ПО получили образование по специальностям, связанным с ПО [80]. Почти половина получивших научные степени в области ПО сначала имели степень бакалавра по каким-то другим специальностям. Еще 20% всех работников отрасли ПО имели степени в таких науках, как математика, инженерия, английский язык, история, философия. Остальные 40% закончили университет или колледж, но не получили степени по итогам четырехгодичного обучения.

Таблица 7.1. Образовательный уровень разработчиков ПО[1]

Самый высокий достигнутый уровень образования	Удельный вес разработчиков ПО, %
Высшее образование или его эквивалент, или более низкий уровень	11,8
Колледж, без присуждения степени	17,2
Диплом младшего специалиста	11,0
Степень бакалавра	47,4
Ученая степень выше бакалавра	12,8

На сегодняшний день университеты США присуждают ежегодно около 35 тысяч степеней в сфере компьютерной науки и по смежным дисципли-

[1] *Occupational Outlook Handbook 2002-03 Edition*, Bureau of Labor Statistics, 2002.

нам,[1] тогда как каждый год создается около 50 тысяч рабочих мест для разработчиков ПО.

Вывод из этих статистических данных таков: огромная масса разработчиков не получает систематического образования в компьютерной сфере, и совсем мало среди них специалистов в такой области, как инженерия ПО. Знания эти разработчики получили в процессе работы или самообразования. Формирование стройной системы образования в области инженерии ПО открывает серьезную возможность повысить уровень практической разработки ПО.

Перспективы занятости

Всего в отрасли ПО в США сегодня занято около 2 миллионов человек. Как видно из табл. 7.2, рабочие места распределены между учеными-компьютерщиками, программистами, системотехниками, сетевиками и инженерами ПО. (Некоторые из названий специальностей в официальной статистике могут показаться устаревшими, но в них включены современные специальности в области ПО).

Таблица 7.2. Структура занятости в отрасли программного обеспечения [55]

Специальность	**Число занятых в США**
Ученые-исследователи в области компьютерных и информационных технологий	28 000
Программисты	585 000
Инженеры прикладного ПО	380 000
Инженеры системного ПО	317 000
Системные аналитики	431 000
Аналитики сетевых систем и передачи данных	119 000
Другие компьютерные специальности	203 000
Всего	**2 063 000**

Перспективы занятости разработчиков ПО в США прекрасные. Согласно данным Бюро статистики занятости (The Bureau of Labor Statistics), услуги в области компьютеров и обработки данных станут самой быстрорасту-

[1] National Center for Education Statistics, *2001 Digest of Educational Statistics*, Document Number NCES 2002130, April 2002.

щей отраслью в 2000–2010 гг., причем предполагаемый рост за этот период составит 86%. По имеющимся оценкам, в инженерии ПО будет наблюдаться самый быстрый рост занятости. Он должен происходить по всем специальностям, связанным с компьютерами.[1]

Такой же взрывной рост числа разработчиков ПО, как в США, прогнозируется во всем мире. Предполагаемый рост числа занятых иллюстрирует табл. 7.3.

Таблица 7.3. Статистика рабочих мест разработчиков ПО во всем мире [67]

Год	Количество программистов
1950	100
1960	10 000
1970	100 000
1980	2 000 000
1990	7 000 000
2000	10 000 000
2010	14 000 000
2020	21 000 000

При сохранении разрыва в 15 тысяч между количеством ежегодно создаваемых рабочих мест и количеством присуждаемых степеней бакалавра спрос на программистов в США должен сохраняться стабильно высоким, по крайней мере в течение ближайших нескольких лет, несмотря на циклические колебания рынка. Нехватка квалифицированных специалистов постоянно преследует отрасль ПО, начиная с середины 60-х годов XX века [24]. По многим показателям (оплата труда, премии, обстановка, стресс на рабочем месте, надежность работы и др.) рабочие места в сфере ПО оцениваются очень высоко [73]. Программистам хорошо известно, что, несмотря на привлекательность рабочих мест, конкуренция не слишком высока.

Герои и узурпаторы программирования

Дефицит квалифицированного персонала в сочетании с распространенным обычаем ставить слишком оптимистичные сроки подготавливает почву к появлению героя. Программисты-герои берутся за слож-

[1] *Occupational Outlook Handbook 2002-03 Edition*, Bureau of Labor Statistics, 2002.

ные заказы и пишут горы программ. Они работают сутками сверхурочно. Они незаменимы в проектах, где они заняты. Успех, как может показаться, держится на их плечах.

Руководители проектов и любят, и опасаются программистов-героев, умных, темпераментных и иногда слегка самоуверенных, потому что не видят способа закончить проект без их помощи [4]. Найти им замену в условиях ограниченного рынка труда вряд ли представляется возможным.

К несчастью, реальность состоит еще и в том, что кроме программистов-героев, способных на подвиги в написании монументальных программ, есть программисты, выступающие в роли источника многочисленных бед, просто не умеющие нормально работать с коллегами. Они прячут исходный код и информацию о структуре программы, отказываются участвовать в технических сверках и следовать стандартам, принятым группой разработчиков. Итог их деятельности – невозможность дать другим членам группы сделать потенциально ценный вклад в общее дело. Очень многие программисты-герои оказываются вовсе не героями, а примадоннами-узурпаторами программирования.

Героические усилия индивидуалов могут способствовать успеху проекта, но коллективные усилия, как правило, дают больше, чем достижения отдельных участников проекта. Проведенное в IBM исследование показало, что программист в среднем тратит лишь 30% своего рабочего времени, работая в одиночку [87]. Все остальное время он работает вместе с коллегами, заказчиком или интерактивно. По результатам исследования 31 проекта ПО самый большой вклад в общую производительность дает слаженность группы разработчиков [77]. Индивидуальные способности тоже влияют на производительность, но все же меньше.

Многим нравится браться за проекты, которые требуют напряжения всех сил и способностей. Те разработчики, которые могут испытать свои возможности, следовать разумной практике разработки ПО и все же сотрудничать со своими коллегами, и есть настоящие программисты-герои.

Культ личности

При внимательном рассмотрении многие названные аспекты стереотипа личности программиста оказываются верными. Из-за нехватки персонала многие работники (и программисты-герои, и обычные сотрудники) вынуждены работать сверхурочно, и поэтому у них остается меньше времени на самообразование и профессиональный рост. Получается

замкнутый круг: нельзя внедрить передовые методики, если нет времени на обучение и образование, а этого времени не остается, если не применять передовые методики.

На самом деле усилия, направленные на повышение профессионализма разработки ПО, говорят о старении программистов. Чем дольше существует отрасль ПО, тем больше средний возраст занятых в ней будет приближаться к среднему возрасту работающего населения. Огромные личные жертвы, допустимые для молодых двадцатилетних сотрудников, труднее оправдывать, когда они заводят семью, детей, становятся владельцами дома и им уже за 30, 40, 50 и даже 60 лет. По мере старения сегодняшнего контингента программистов практикуемый подход к разработке ПО на основе героических усилий отдельных работников может естественным образом уступить место подходам, больше полагающимся на разумный, нежели на напряженный труд. Многие разработчики ПО возьмут на вооружение методики, которые позволяют завершить проекты к обещанному сроку и все-таки быть дома к ужину.

ГЛАВА ВОСЬМАЯ

Формирование сознательного отношения к ПО

Тот, кто начал свой путь уверенно, закончит сомнениями, тот же, кто сомневается в начале пути, в конце идет уверенным шагом.

Френсис Бэкон

В 1970 г. Чарльз Рейч (Charles Reich) опубликовал свой бестселлер «Молодая поросль Америки» [116], в котором выделил три типа восприятия, или сознания, названные им сознание 1, сознание 2 и сознание 3.

Сознание 1 («Con 1», по Рейчу) – это ментальность первопроходца-первооткрывателя. Люди, действующие на этом уровне сознания, придают большое значение независимости и самоудовлетворению. Они не терпят приказов других людей. В высокой степени самодостаточны и надеются на себя. Рейч считал, что сознание 1 доминировало в психике американцев в первые века существования страны, и эта ориентация на самостоятельность была существенным фактором развития Америки.

Сознание 2 – ментальность «серого фланелевого костюма», корпоративного человека. Люди, функционирующие на этом уровне сознания, понимают, что надо ладить с другими людьми и играть по правилам. Они считают, что правила нужны и полезны для общества и все должны их соблюдать. По Рейчу, сознание 2 стало превалировать над сознанием 1 во второй половине двадцатого века.

Сознание 3 – это ментальность просвещенной независимости. На этом уровне сознания люди функционируют, опираясь на принципы и не обращая особого внимания на правила, которые доминируют в сознании 2,

но делают это без эгоцентризма, доминирующего в сознании 1. К моменту опубликования своего труда Рейч утверждал, что время сознания 2 прошло, и считал, что сознание 3 на подъеме и скоро заменит сознание 2.

Хотя при опубликовании книга Рейча нашла отклик в обществе, время ее не пощадило. В 1999 г. читатели журнала *State* голосованием назвали ее «самой глупой книгой» двадцатого века. Сознание 3 по Рейчу оказалось нирваной хиппи, а «молодая поросль», которую предсказывал Рейч, превратилась в общенациональное движение к культуре хиппи в 60–70-х гг. прошлого века – психоделические наркотики, расклешенные брюки и т. д. С уходом в забвение культуры хиппи в 80-х годах XX века туда же канула и достоверность предсказаний Рейча.

Нет удовлетворения

Возможно, политические предсказания Рейча не выдержали проверки временем, но его классификация сознания дает полезную основу для модели отрасли ПО сегодня.

Сознание 1 в ПО ассоциируется с ориентацией на *самодостаточность*. Эксперты отрасли часто называют разработчиков ПО, работающих на этом уровне сознательности, индивидуалистами, программистами-ковбоями, одинокими рейнджерами и примадоннами. Для таких разработчиков характерно пренебрежение к чужим идеям. Они любят работать в одиночку и не любят следовать стандартам. Среди них процветает синдром «изобретено не мной».

Преимущество сознания 1 в том, что не нужно широкой подготовки и обучения, а подход «одинокого волка» пригоден в условиях, когда небольшое количество программистов работают независимо над небольшими проектами. Недостаток этого уровня сознания заключается в его плохой приспособляемости к тем проектам, где необходима именно группа программистов, а не отдельные личности. Поэтому ценность сознания 1 ограничена лишь самыми мелкими проектами.

Сознание 2 в ПО ассоциируется с упором на *правила*. Многие разработчики ПО в конечном итоге осознают ограниченность сознания 1 и приходят к пониманию преимуществ работы в группе. Со временем усваиваются правила, позволяющие координировать свою работу с работой коллег. Некоторые группы разработчиков методом проб и ошибок вырабатывают свои неформальные правила, и такие группы могут работать очень эффективно. Другие перенимают уже готовую методологию. Иногда правила соз-

дают консультанты, примером тому может служить классическая методология «17 трехзвенных кольцевых сшивателей» («17 three-ring-binders»). В других случаях правила берутся из книг, например из «The Rational Unified Process: An Introduction» [75],[1] из книг серии «Экстремальное программирование» [8], из моей книги «Software Project Survival Guide» [84].[2] Для разработчиков на этом уровне характерна сосредоточенность на деталях соблюдения правил. Они спорят о том, какая интерпретация правил правильнее, и уделяют большое внимание «следованию методологии».

Сознание 3 в ПО ассоциируется с упором на *принципы*. На этом уровне разработчики понимают, что правила любой сложившейся методологии в лучшем случае лишь приближаются к принципам. Эти приближения, возможно, применимы в большинстве случаев, но не всегда. Требуется широкое образование и интенсивное обучение, чтобы привести разработчика к принципам сознания 3, лежащим в основе эффективной разработки ПО, но такое образование и обучение нелегко найти. Однако, получив нужное образование и подготовку, разработчик вооружается полным арсеналом инструментов инженерии ПО для достижения успеха в широком спектре проектов.

В то время как подход сознания 2 заключается в повторяемом применении методов и процедур, опора сознания 3 на принципы требует разумного выбора и творчества. На уровне сознания 2 разработчика можно обучить применению лишь одного подхода. Если выбран разумный подход, то разработчик может, опираясь на ограниченный объем знаний, использовать его во многих проектах. Разрыв между уровнями сознания 2 и 3 кажется незаметным в сфере применимости сознания 2. Однако разработчик уровня сознания 2 плохо подготовлен к достижению успеха в проектах, не подпадающих под конкретную методологию, которой он обучен. По существу разработчик на уровне сознания 2 являет собою индивидуальный пример «культа карго», описанного в главе 3, то есть применения методики без глубокого понимания причин ее эффективности, что приводит к непостоянству результатов работы в проектах.

---

[1] Ф. Кратчен «Введение в Rational Unified Process», 2-е издание, Вильямс, 2002.

[2] С. Макконнелл «Остаться в живых! Руководство для менеджера программных проектов», СПб.: Питер, 2006.

Возлюби тех, с кем работаешь

Отрасль ПО имеет долгую историю попыток изобрести панацею – средство, пригодное на все случаи жизни. В конечном итоге все такие «универсальные» методологии отвергались. Среди них и подход к ПО на уровне сознания 2, который, разумеется, не срабатывает вне узко заданной области применения именно оттого, что это сознание 2. Мир ПО слишком разнообразен, чтобы его можно было охватить каким-то набором правил.

Для примера сравним набор правил, который применялся бы при разработке ПО для управления биостимулятором сердечных ритмов и программы контроля товаров в магазине видео. Если ошибка в ПО приведет к пропаже одного видеодиска из тысячи, прибыль магазина уменьшится на доли процента, поэтому ущерб пренебрежимо мал. Если же ошибка приводит к отказу одного из тысячи биостимуляторов, то возникает настоящая проблема. Вообще говоря, продукты массового спроса должны разрабатываться более тщательно, чем продукты ограниченного применения, а если, важную роль играет надежность, то продукт следует разрабатывать более внимательно.

Различие в типах ПО требует дифференцированных технологий разработки. Методики, которые можно счесть излишне строгими, бюрократическими и обременительными при разработке ПО для магазина видео, могут показаться безответственно торопливыми и неаккуратными или даже небрежными, когда речь идет о встроенном ПО управления биостимулятором. Разработчик на уровне сознания 3 будет использовать разные методики при разработке ПО биостимулятора и системы контроля складских запасов магазина видео. Разработчик, действующий на уровне сознания 2, попытается применить универсальную методологию «на все размеры» к обоим продуктам. При этом существует вероятность, что она не будет подходить точно «по размеру» ни к одному из продуктов.

Насколько вы опытны?

Рейч определил три уровня сознания как отражение различных эпох. Мне они представляются как три последовательных шага к личностной зрелости в инженерии ПО. Большинство разработчиков начинают с уровня сознания 1, в конце концов переходя к сознанию 2. В некоторых условиях сознание 2 обеспечивает эффективность работы и не нуждается

в дальнейшем продвижении. Однако во многих ситуациях необходим переход на уровень сознания 3.

Изложенные в учебниках методологии уровня сознания 2 являются естественным путем для разработчиков уровня сознания 1, которые еще слабо ориентируются в широком диапазоне методик разработки ПО. Конкретные детали технологий на основе правил, вероятно, не очень важны. Разработчикам, стремящимся перейти от сознания 1 к сознанию 2, достаточно сделать первый шаг и уйти от хаоса полностью неуправляемых проектов. Надо освоить набор правил и накопить некоторый опыт их применения, перед тем как попытаться перейти на уровень сознания 3. На этом уровне появляется достаточное понимание динамики проектов ПО, чтобы уметь отойти от правил, когда это необходимо. Весь процесс – это часть естественного развития от ученика к подмастерью и далее к мастеру.

ГЛАВА ДЕВЯТАЯ

Формирование сообщества

Если вместо сохранения и использования старого человек целеустремленно пытается продвинуться к новым открытиям, одерживать победы над Природой как работник, а не выигрывать как спорщик в дискуссиях с враждебной критикой, реально получать ясные и четкие знания, а не выдвигать привлекательные и вероятные теории, то мы приглашаем его в наши ряды как настоящего ученого мужа.

Френсис Бэкон

В 1984 г. я получил свою первую работу на полный рабочий день в качестве программиста, став аналитиком в консультационной фирме с пятью сотрудниками. Зарплата была приличная. И диетическая пепси-кола бесплатно. Кроме того, я начал работать над проектами, связанными с ПК фирмы IBM, что было намного интереснее работы с ЭВМ коллективного доступа в институте.

Проекты были масштабнее институтских и длились от нескольких дней до месяца. Я научился кое-чему, чего не умел в институте: координировать свою работу с коллегами; терпеть начальство, постоянно меняющее технические требования к проектам; работать с заказчиками, которые зависели от ПО и предъявляли претензии, если оно работало не так, как им было нужно.

Моя вторая работа программистом – участие в крупном аэрокосмическом проекте на базе большой ЭВМ коллективного доступа – была типичным госпроектом, погрязшим в бумаготворчестве. Неэффективность его была просто поразительной. Я убежден, что три-четыре программиста с моей прежней работы написали бы за три месяца то, что группа из трид-

цати разработчиков этого проекта сделала за три-четыре года. (Я по-прежнему считаю, что это, скорее всего, так и есть).

У одного из моих коллег по проекту была книга Никлауса Вирта (Niklaus Wirth) «Algorithms + Data Structures» [143],[1] и его считали настоящим идеологом проекта – скорее по внешним признакам, потому что он читал эту книгу. Мне не очень нравилось это занятие, и я с радостью забывал о программировании, как только уходил с работы.

Поработав в аэрокосмической компании, я снова вернулся в небольшую фирму, где был единственным постоянным программистом. Я приступил к работе над захватывающим проектом примерно на год по «упаковке» DOS в красивую обертку. Программа на языке C должна была выжимать максимум из тогдашних ПК. Пепси-колой меня бесплатно не поили, но я был доволен, что снова работал на ПК. Единственная загвоздка была в том, что новый проект опять пробудил во мне страсть к программированию, но вокруг не было никого, кто разделял бы мой энтузиазм.

К тому времени я проработал около трех лет в отрасли ПО на постоянной основе, но, за исключением руководств по ЭВМ и справочников по языкам программирования, не читал учебников по программированию и не подписывался на специализированные журналы. (Хотя и приобрел книгу «Algorithms + Data Structures».)

Небольшая фирма, в которой я работал, подавала себя как программистская «Команда А». На собеседовании при приеме на работу мне сказали, что у фирмы есть множество собственных секретных разработок, которые позволяли ей считаться «Командой А». Когда я приступил к работе, мне захотелось узнать эти самые «секреты фирмы». И тогда мой начальник вручил мне книжку Филиппа У. Метцгера об управлении проектами ПО [89, 90]. Я сразу же ее прочитал и был поражен тем, что автор, по-видимому, прошел во многом через то же, что и я. Планирование моего годичного проекта DOS вызывало трудности. Книжка Метцгера решила многие проблемы, и я взял ее за основу при планировании оставшейся части проекта.

Сразу после прочтения книги Метцгера мне попалась книга Эда Йордона (Ed Yourdon) и Ларри Константина «Structured Design» [144]. Пролистав ее, я нашел объяснение трудностям, с которыми мне пришлось столкнуться при проектировании программы DOS. По рекомендации авторов я перевернул иерархию вызовов подпрограмм, и все стало на свои места. Я начал понимать, что, наверное, полезной для меня информации гораздо

[1] Н. Вирт «Алгоритмы и структуры данных», СПб.: Невский диалект, 2005.

больше, чем я предполагал. В то время мне смутно помнились какие-то буквы, которые произносили мои преподаватели: A-C-M и I-E-E-E. У меня не было диплома профессионального программиста, но я все равно решил попытаться стать членом этих организаций. Я подписался на журналы *Communications of the ACM*[1], *IEEE Computer*[2] и *IEEE Software*[3]. Последнее издание быстро стало моим любимым. Я нашел статьи, посвященные вопросам, решение которых позволяло более эффективно выполнять работу: как помочь заказчикам принять решение по техническим требованиям, как контролировать сложность крупных проектов, как создавать обозримые программы, как координировать работу нескольких программистов и т. д. Статьи не были такими расплывчатыми, как в некоторых читаемых мной популярных журналах, и я посчитал, что они могли быть полезны в моей дальнейшей карьере.

Это был настоящий водораздел в моем развитии как разработчика ПО. До вступления в ряды читателей журнала *IEEE Software* я рассматривал программирование просто как работу. Я работаю – мне платят. Я пришел домой с работы и перестал думать о программировании. Став читателем журнала, я начал осознавать, что я, хоть и работаю один, не просто программист-одиночка, а член сообщества разработчиков ПО, которых волнуют проблемы их профессии и которые тратят свое время, чтобы поделиться опытом на благо других.

Можно предположить, учитывая огромный разброс в уровне образования и профессионализма в отрасли ПО, что на сегодняшний день пока не сложилось серьезное сообщество разработчиков ПО, однако мне кажется, что этот разброс делает задачу его формирования еще более насущной. В любом сообществе есть более или менее профессиональные члены, и оно должно отвечать нуждам как молодых, так и зрелых практиков. Оно должно учитывать запросы людей с прочной образовательной основой в компьютерной науке, инженерии ПО и смежных областях и потребности программистов-самоучек, ученых, инженеров, бухгалтеров, учителей, врачей, юристов и других, которые, как оказалось, зарабатывают на жизнь написанием программ, хотя сознательно так и не приняли решения стать разработчиками или инженерами ПО.

---

[1] См. *www.acm.org*.

[2] См. *www.computer.org/computer*.

[3] См. *www.computer.org/software*.

Профессиональные ассоциации, такие как компьютерное общество IEEE, являются важным атрибутом зрелой профессии. Они открывают возможность профессионалам со сходным мышлением собираться вместе, обмениваться идеями в личном общении, через статьи, группы по интересам и на конференциях. Профессиональные организации поддерживают многообразные структурированные способы обмена хитростями и приемами профессии, которые необходимы инженерам ПО, чтобы действительно быть в Команде А.

ГЛАВА ДЕСЯТАЯ

Архитекторы и строители

Инженеры разрабатывают проекты.
Производители реализуют их в продукте.

Терри Магиннис (Terry Maginnis)

В сформировавшихся сферах деятельности профессии стратифицированы и специализированы. В строительной индустрии архитекторы и инженеры создают проекты, а генеральный подрядчик по этим проектам строит. Обычно генеральный подрядчик передает часть работ специализированным субподрядчикам, монтажникам, сантехникам, электрикам, устроителям ландшафта. В отрасли ПО происходит свое разделение на архитекторов ПО и строителей и специализация по монтажу, сантехнике, электрике и т. д.

Стратификация профессии

В других отраслях профессиональным инженерам обеспечивается поддержка инженеров-технологов и техников. Национальный институт сертификации инженерных технологий (The National Institute for Certification in Engineering Technologies) предлагает пять классификационных групп. Аналогичное вертикальное разделение наблюдается в медицине: врачи, ассистенты, фельдшеры, медсестры и санитарки. В юриспруденции есть адвокаты, ассистенты и секретари адвокатов. Все сложные виды деятельности стратифицируются.

Как показано на рис. 10.1, по мере зрелости инженерии ПО профессия будет вертикально разбиваться на специальности, требующие большего или меньшего объема образования и подготовки. Кто-то из разработчиков

получит общее образование в профессиональной инженерии ПО, другие обучатся программированию как ремеслу. Некоторые из получивших профессиональное образование в инженерии ПО пойдут дальше и будут лицензированы по профессиональной инженерии ПО, но большинство останется на своем уровне. Те, кто получит лицензии, станут аналогом практикующего врача, а получившие общее образование, но не ставшие полными профессионалами,– ассистентами.

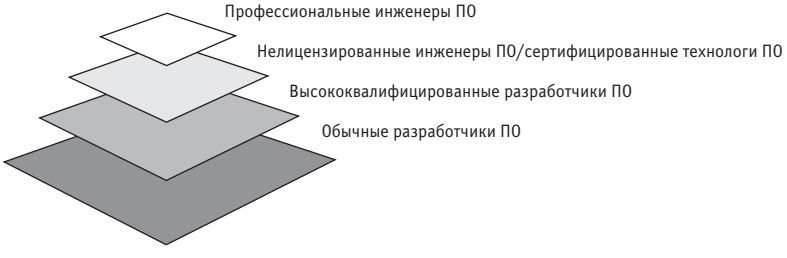

Рис. 10.1. *Отрасль разработки ПО стратифицируется вертикально по разным уровням профессионализма. Наиболее подготовленные и профессиональные работники в целом будут иметь более широкие полномочия, и их труд будет лучше оплачиваться*

У инженеров и технологов ПО – сегодняшних программистов – тоже будут разные свидетельства профессионализма. Технологи будут эквивалентом ассистента или медсестры в медицине, или сертифицированного технолога в промышленности. Одни технологи ПО захотят получить лицензии, другие нет. В общем работа, требующая более высокого уровня образования и подготовки, будет означать большую ответственность и высокий престиж – так же, как и в других профессиях.

Разработчики ПО, получившие высшие квалификационные сертификаты, вероятно, будут получать более высокую зарплату, чем другие разработчики. Как правило, лицо, получившее профессиональный квалификационный сертификат в США, зарабатывает в среднем по крайней мере в полтора раза больше, чем лицо, имеющее лишь степень бакалавра [97]. Степень мастера позволяет зарабатывать в среднем на 25% больше, чем зарабатывают бакалавры. И этот разрыв, вероятно, будет увеличиваться. Бюро статистики занятости утверждает, что с 2000 г. по 2010 г. спрос на специальности, требующие обычной практической подготовки, вырастет на 11%, а на специальности, требующие диплома мастера, – на 23% [100].

Специализация функций

Помимо стратификации специальностей отрасль ПО нуждается в формировании специализации. Большинство занятых в ней специалистов – это мастера широкого профиля (сейчас они архитекторы, а вот становятся хайтек-плотниками, сколачивающими код буквально по строчечке). В сформировавшихся профессиях специализация является важнейшим элементом.

Двадцать пять лет назад Фредерик Брукс предложил тип специализации в инженерии ПО, в соответствии с которым группа разработчиков делилась по принципу хирургической бригады. Один главный программист («хирург») создает почти всю программу, а другие члены бригады группируются вокруг него, исполняя четко определенные вспомогательные роли [21]. Опытный проект в конце 1960-х годов, структурированный по этому принципу, показал невиданную производительность [6], [7], и Брукс пришел к выводу, что подобный тип структуры по принципу бригады хирургов обеспечил такой успех.

Глядя на этот необычный проект с высоты прошедших двадцати пяти лет, думается, что замечательная производительность, возможно, объясняется не конкретным построением группы разработчиков по типу бригады хирургов, а высокой степенью специализации работников в проекте. Исследования других методик инженерии ПО показали, что хорошая подготовка по специальности дает больший вклад в эффективность работы, чем применение конкретных методик.[1]

Сегодня в отрасли ПО складываются две группы специализации: технология и инженерия ПО. Как показывает рис. 10.2, специализация технологов ПО будет основана главным образом на владении определенными технологиями. Различные сертификаты «технологов» уже выдают некоторые компании, такие как Microsoft, Novell, Oracle, и Apple Computers.

---

[1] Брукс высказывает аналогичную мысль на стр. 257 20-го юбилейного издания книги [21] (Ф. Брукс «Мифический человеко-месяц или как создаются программные системы», СПб.: Символ-Плюс, 2000, стр. 237), хотя не увязывает ее с хирургической бригадой. «Я стал настаивать, чтобы даже небольшие группы студентов из четырех человек выбирали себе отдельно менеджера и архитектора ПО. Назначение на роли в таком небольшом коллективе, может быть, и крайность, но я убедился, что это очень удачный ход, способствующий успешной работе даже маленьких коллективов».

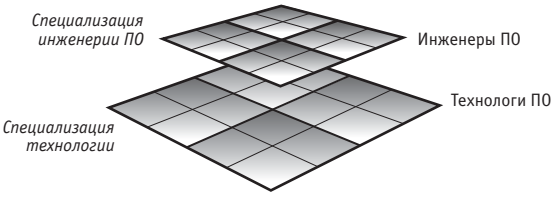

Рис. 10.2. *Помимо стратификации в отрасли ПО развиваются многие различные технологические и инженерные специальности*

Но специализация начинает складываться и в инженерии ПО, и это важная тенденция. По оценкам Кейперса Джоунза, недостаток специализации в инженерии ПО сегодня приводит к низкому качеству, затягиванию сроков и перерасходу смет примерно в 90% организаций-разработчиков ПО в США [64].

Данные табл. 10.1 показывают, что чем больше специалистов по ПО работает в компании, тем больше ощущается необходимость в их ориентации на определенный участок работы, а не просто программирование. В небольшой организации из 10 инженеров ПО все могут выполнять любую работу, или же возможны небольшие различия между разработкой, тестированием и управлением ПО. В крупных компаниях-разработчиках ПО, в которых работает по 10 тысяч человек, не менее 20% работающих должны быть специалистами, а в некоторых организациях доля специализации может доходить до 40%. Оценивая организационные аспекты, Джоунз выявил свыше ста различных специализаций.

Конкретные специальности в таблице получены грубым усреднением. Доля специалистов по отношению к числу работников общего профиля меняется в зависимости от компании и различных схем организации компаний-разработчиков ПО.

Преимущества специализации характерны не только для индустрии ПО. Практикующему сельскому врачу приходится быть врачом широкого профиля, мастером на все руки, но в крупной городской больнице работают сотни специалистов. Профессиональные инженеры сдают экзамен по специальности, как и адвокаты. Специализация – это атрибут зрелости отрасли.

Таблица 10.1. Соответствующие специализации в зависимости от размера компании [64], [66]

Специальность	Число сотрудников ПО				Доля от числа программистов широкого профиля, %
	Менее 10, %	Менее 100, %	Менее 1 тыс., %	10 тыс., %	
Доля специализации	0	10–25	15–35	20–40	–
Архитектура			X	X	1: 75
Контроль конфигурации			X	X	1: 30
Оценка стоимости			X	X	1: 100
Поддержка/сопровождение клиента		X	X	X	1: 25*
Администрирование баз данных		X	X	X	1: 25
Образование и обучение				X	1: 250
Учет функциональных узлов			X	X	1: 50
Человеческий фактор				X	1: 250*
Информационные системы				X	1: 250*
Интеграция				X	1: 50
Обслуживание и расширение	O	X	X	X	1: 4
Измерения			X	X	1: 50
Сеть		X	X	X	1: 50
Приобретение пакетов ПО				X	1: 150
Производительность				X	1: 75
Планирование					1: 250*
Совершенствование процессов				X	1: 200
Обеспечение качества	O	X	X	X	1: 25
Технические требования			X	X	1: 50*

Специальность	Число сотрудников ПО				Доля от числа программистов широкого профиля, %
	Менее 10, %	Менее 100, %	Менее 1 тыс., %	10 тыс., %	
Использование в иных условиях				X	1:100
Стандарты				X	1: 300
Поддержка системного ПО		X	X	X	1: 30
Технические описания	O	X	X	X	1:15
Тестирование	O	X	X	X	1: 8
Разработка инструментария				X	1: 250*

* Данная величина оценена на основе описаний Джоунса, но конкретное значение не приводится.
O – встречается редко, X – регулярно.

Специализации в коллективе

Отдельным проектам специализация так же необходима, как и организациям. В моей компании внедрена организация проектов на основе свода знаний Swebok, описанного в главе 5. Даже к самым небольшим проектам привлекаются следующие специалисты:

- руководитель написания программ;
- руководитель проектирования ПО;
- руководитель планирования и отслеживания версий;
- бизнес-менеджер проекта;
- руководитель обеспечения качества;
- руководитель разработки требований.

В большинстве проектов эти обязанности выполняются по совместительству, однако опыт показал, что полезно иметь конкретное лицо, отвечающее за интересы проекта в каждой из перечисленных сфер. Специализация необходима даже в тех проектах, где занято от пяти до десяти человек. Более подробно об этих областях руководства, включая рекомендуемые программы обучения для каждой из них, можно узнать на сайте компании Construx по адресу *www.construx.com/profession*.

Время покажет

Предопределен ли вывод о растущей стратификации и специализации, или же этот прогноз будет выглядеть нелепым лет через двадцать? Как подсказывает мой волшебный кристалл, через достаточный промежуток времени разработка ПО станет такой же специализированной и стратифицированной областью, как и другие более зрелые отрасли. Это уже произошло в медицине, юриспруденции, профессиональных инженерных дисциплинах. Мой волшебный кристалл, однако, не говорит, сколько времени потребуется для формирования специализации и стратификации инженерии ПО – десять, двадцать лет или больше.

ГЛАВА ОДИННАДЦАТАЯ

Программист пишущий

Чтение нужно не для того, чтобы возражать и опровергать,
не для того, чтобы соглашаться и верить на слово,
не ради разговоров и досужей болтовни,
но чтобы размышлять и оценивать.

Френсис Бэкон

В 1987 г. Фредерик Брукс указывал, что «разрыв между лучшими образцами разработки ПО и средним уровнем огромен, возможно, он даже больше, чем в любой другой инженерной дисциплине. Поэтому распространение передового опыта чрезвычайно важно» [20]. В развитие мыслей Ф. Брукса Совет по компьютерной науке и технологиям (Computer Science and Technology Board) в 1990 г. заявил, что самые крупные достижения в обеспечении качества и производительности разработки ПО будут сделаны в результате распространения эффективных методик – кодификации, унификации – и распространения накопленных знаний в форме справочников по инженерии ПО [122].

Кто же напишет эти справочники?

В 1837 г. Ральф Уолдо Эмерсон (Ralph Waldo Emerson) выступил с речью, которая стала известна под названием «The American Scholar» (Американский мыслитель). Спустя почти сто семьдесят лет мне кажется, что в ней есть ответ на вопрос, кто должен написать справочники по инженерии ПО.

По мере формирования издательств, выпускающих литературу о ПО, большинство книг, посвященных этой теме, создаются шестью категориями авторов:

- разработчиками ПО, недавно прекратившими активную деятельность;

- профессорами университетов;
- преподавателями семинаров;
- консультантами;
- разработчиками «мозговых центров»;
- разработчиками действующего ПО.

Каждая из этих групп может внести ценный вклад в общее дело, и было бы ошибкой не замечать какую-либо из них. Недавно отошедшие от дел специалисты привносят в свои книги годы опыта, глубокое понимание и размышления. Профессора университетов информируют о новейших исследованиях и разработках. У преподавателей семинаров имеется возможность испытать свои наработки перед сотнями обучающихся, прежде чем опубликовать их в книгах. Консультанты ежегодно встречаются с десятками клиентов и излагают свои соображения на основе знания широкого спектра эффективных и неэффективных методик разработки ПО. Разработчики в «мозговых центрах», таких как Xerox PARC, AT&T Labs и других, причастны к созданию лучших образцов инженерии ПО. Однако я считаю, что главную ношу написания таких книг должны нести разработчики действующего ПО.

В своем эссе «Американский мыслитель» Эмерсон проводит различие между «созерцателем» и «человеком мыслящим» (это синоним «американского мыслителя»). Единственная функция созерцателя состоит в том, чтобы думать. Ощущение жизни приходит к нему через книги, статьи, описания живого мира, созданные другими людьми. Мыслитель же, наоборот, активно действует в мире, вовлечен в профессию или ремесло, время от времени останавливаясь для размышлений. Мыслитель твердо ориентирован на действия: «Настоящий мыслитель сожалеет о каждой потерянной возможности действовать как о пропавшей силе. Это сырье, из которого мыслитель создает свои великолепные произведения». Эмерсон утверждал, что непосредственный опыт критически важен для рождения гения, которым способен стать только мыслитель, а не созерцатель.

Далее Эмерсон говорит, что активность восприятия необходима для понимания того, что об этом мире написали другие мыслящие люди, и для усвоения прочитанного. А «когда мозг вооружен действием и воображением, любая прочтенная страница вызывает многообразные ассоциации. Каждое предложение несет удвоенную силу, а мысль автора широка, как сам мир».

Если читатель, «не вооруженный действием и воображением», мало что вбирает из прочитанного, то писатель, подобный читателю, мало что вкла-

дывает в написанное. Не сопровождаемая действием мысль никогда не породит истину. Читатель сразу ощущает, в чьих словах бурлит жизнь, а чьи слова безжизненны. «Я научился догадываться, насколько деятелен говорящий, по яркости или тусклости его речи. Жизнь лежит за нашей спиной, как карьер, из которого мы берем глину для плитки и кирпичей сегодняшней кладки. Книги и учебные заведения лишь копируют язык, создаваемый в поле и в мастерской».

Я называю неумение простых созерцателей достоверно излагать мысли «синдромом Фенимора Купера» – американского писателя эпохи освоения Америки европейцами, очарованного американскими индейцами и написавшего несколько новелл и романов о них, в том числе «Охотник за оленями» и «Последний из могикан». Произведения Купера позже едко высмеял великий американский писатель и юморист Марк Твен, назвав их безнадежными фантазиями [135].

В сцене из «Охотника за оленями» Купер описывает, как шесть индейцев вскарабкались на нависающую над рекой ветвь, дожидаясь, когда вверх по течению будут тянуть на канате баржу. Индейцы выжидали момент, когда можно будет спрыгнуть на крышу рубки, расположенной посредине баржи длиной около 27 и шириной около 5 метров. Один из них промахнулся и упал на корму, а остальные и вовсе попадали в воду.

Марк Твен был шкипером на речном судне и высмеял Фенимора Купера за беспомощное описание предмета, который сам знал в деталях. Во-первых, ветка вряд ли могла выдержать шестерых взрослых мужчин. Далее, баржа в треть описанных Купером размеров с трудом могла бы проходить повороты и излучины реки – она застряла бы на первом повороте. Что касается прыжка индейцев на крышу, то вверх по течению баржу могли бы тянуть канатом с максимальной скоростью около мили в час, так что у индейцев было почти полторы минуты, чтобы прыгнуть на баржу, и около минуты, чтобы спрыгнуть с ветви на крышу рубки, – достаточно времени, чтобы не рассчитывать до секунды момент прыжка. Тем не менее «куперовские» индейцы все равно ухитрились промахнуться. Возможно, они расслабились, когда увидели, что ширина реки в том месте, где баржа проплывала под ними, лишь на пару метров больше ширины самой баржи – можно было сэкономить силы и не карабкаться на дерево, а просто запрыгнуть на баржу с берега, но, как заметил Марк Твен, писатель не позволил своим индейцам прыгать с берега, поэтому в их неудаче виноват Купер, а не они.

На 17-й международной конференции по инженерии ПО Дэвид Парнас заметил, что доклады, признанные ведущими на предыдущих конферен-

циях, вероятно, вовсе не были таковыми [106]. Думается, что частично это связано с «синдромом Фенимора Купера». Возможно, эти доклады оказали влияние на исследователей, но они были бесполезны для практиков, так как описываемые методологии были настолько же реальными, насколько реальными были индейцы у Купера.

Практикующие разработчики ПО так же скептически настроены в отношении книг по инженерии ПО, как Марк Твен по отношению к куперовским творениям. Справочники по разработке ПО считаются теоретизированием, пригодным лишь для небольших проектов, трудно адаптируемыми, неэффективными и неполными. Авторы книг о ПО частенько вздыхают, что средний разработчик покупает меньше одной книги по данному предмету в год, однако причины этого, похоже, кроются не за семью печатями. Разработчики будут покупать книги, «вооруженные практикой и творчеством», а не те, которые, подобно «куперовским» индейцам, бьют мимо цели.

Марк Твен утверждал, что лучшие приключенческие романы написаны людьми, зорко подмечавшими детали, реально жившими на передовых рубежах переселенцев. А я убежден в том, что лучшие книги о ПО будут основаны на работе программистов, которые только что завершили реализацию проектов ПО. Применяя метафору Эмерсона о плитках и кирпичах к созданию справочников ПО, можно сказать, что они должны быть плодом трудов «программиста пишущего», активно участвующего в проектах написания ПО, размышляющего о своей работе и пишущего о ней.

Если вы активно работающий программист, я призываю вас писать о ваших размышлениях и находках. Если вы работали в проекте и получили полезный опыт, напишите об этом – неважно, касается ли это написания программ, обеспечения качества, более эффективного управления проектом или даже темы в разработке ПО, у которой еще нет названия. Пошлите ваши заметки в журнал или развивайте свои идеи до формата книги. Если ваши мысли глубже, чем ваши описания, привлеките консультанта, преподавателя семинаров, университетского профессора или более искусного писателя в качестве соавтора. Не надо беспокоиться, что познанное вами неприменимо к другим проектам. Как говорил Эмерсон: «Успех идет по следам каждого шага. Потому что инстинкт безошибочен и побуждает рассказать ближнему о своих мыслях. Потом становится понятным, что, проникая в тайны своего ума, человек познает секреты мышления других людей. Все глубже познавая собственные, самые глубокие мысли, человек с удивлением обнаруживает, что это и есть самая приемлемая, общепринятая и универсальная истина» [123].

ЧАСТЬ ТРЕТЬЯ

Организационный профессионализм

ГЛАВА ДВЕНАДЦАТАЯ

Золотая лихорадка ПО

В богатстве проявляется порок, в лишениях – добродетель.

Френсис Бэкон

Корень предрассудков в том, что люди замечают попадания в цель, но не видят промахов.

Френсис Бэкон

В январе 1848 г. Джеймс Маршалл (James Marshall) нашел золото в Калифорнии в реке Американ возле мельницы, которую он строил для Джона Саттера. Сначала Маршалл и Саттер посчитали самородки величиной с горошину помехой: они боялись, что внимание, которое, несомненно, привлекла бы весть о находке, повредит планам Саттера по созданию агроимперии. Но за несколько месяцев известие о золоте распространилось, и к 1849 г. тысячи мужчин и горстка женщин со всего мира подхватили «золотую лихорадку». Они бросились в Калифорнию в поисках состояния, и этот процесс получил название Калифорнийской золотой лихорадки. Поток золотоискателей на Запад создал новую экономику, подпитываемую предпринимательством с высоким риском и мечтами быстро разбогатеть. В дни этого бешеного поиска лишь несколько счастливчиков из первопроходцев, приехавших в 1849 г., смогли превратить свою мечту в реальность, но она живет... во многих современных компаниях ПО и отдельных разработчиках.

Уникальность золотой лихорадки в Калифорнии состояла в том, что золото находили в реках, а не в твердой породе. Поэтому поначалу любой старатель с цинковым тазом и духом предпринимателя мог намыть целое состояние. Но к середине 1849 г. большая часть легко добываемого золота бы-

ла найдена, поэтому рядовому старателю приходилось по десять часов стоять в ледяной воде, поднимая грунт, просеивая и промывая его. Со временем этот каторжный труд приносил все меньше и меньше золота. Вплоть до 1850-х годов изредка еще случались крупные находки, но их хватало лишь на новостные сенсации, вдохновлявшие на продолжение стараний. Большинство старателей не находили золота, но продолжали искать годами.

После первых дней золотой лихорадки старателям пришлось прибегнуть к более изощренным методам добычи. К началу 1850-х годов старатель-одиночка уже не мог обработать свой участок. Ему нужна была помощь других людей и технологий. Сначала золотоискатели стали стихийно объединяться в бригады, чтобы построить дамбы, отвести в сторону русла рек и добывать золото. Но вскоре потребовались более капиталоемкие методы, и бригады старателей заменили компании. К середине 1850-х годов большинство оставшихся золотодобытчиков были работниками корпораций, а не индивидуальными предпринимателями.

Золотая лихорадка в ПО

Пришествие новых технологий разработки ПО часто означает начало очередной «золотой лихорадки ПО». Организации и отдельные предприимчивые люди бросаются в новую технологическую область в надежде, что немного упорного труда, – и они создадут продукт, который сделает их богатыми. Лично я наблюдал это, когда появились ПК и MS DOS, когда состоялся переход от DOS к Windows и когда вычислительные системы распространились в Интернете. Можно не сомневаться, что очередной приступ не за горами.

Для разработки ПО по принципу «золотой лихорадки» характерны методики с высоким риском и высокой отдачей. Лишь несколько компаний смогли закрепиться на конкурентном рынке на заре развития новых технологий ПО, и многие золотые самородки новых технологий (удачные новые продукты), казалось, валялись под ногами, дожидаясь, что кто-нибудь, вооруженный нужным набором инициативы и инновационного мышления, поднимет их. Разработчики ПО, по аналогии со старателями в Калифорнии, пытались «застолбить участки», опередив других. Обычно они работали в паре где-нибудь в гараже, как легендарные дуэты Билла Гейтса и Пола Аллена из Microsoft, Стива Джобса (Steve Jobs) и Стива Возняка (Steve Wozniak) из Apple Computers или Боба Френкстона (Bob Frankston) и Дэна Бриклина (Dan Bricklin) из VisiCalc.

Практические методики, применяемые разработчиками ПО, заболевшими «золотой лихорадкой», обычно ассоциируются со штурмовщиной, а не с инженерией: неформализованные процессы, долгие часы работы, отсутствие достаточного количества документации, едва проклюнувшееся обеспечение качества – другими словами, разработка по принципу «напишем и исправим» отдельными программистами-героями. Такие методики не требуют серьезного обучения, у них невысокие издержки, но для подобных проектов очень высок риск неудачи.

Шансы напасть на золотую жилу во время лихорадки ПО примерно такие же, как и во времена Калифорнийской золотой лихорадки, – на каждую удачу приходятся сотни провальных проектов. Но эти небольшие неудачи не так интересны, как крупные успехи, поэтому о неудачах не слышно. Двое упорно работающих парней, от которых ускользает золото, – не очень горячая новость, если только у них в гараже не спрятано что-то интересное.

Как и тогда в Калифорнии, проекты ПО, осуществляемые на основе героизма разработчика во время «золотой лихорадки», настолько прибыльны в случае удачи, что это убеждает разработчиков ПО в практичности рискованных методов разработки, поэтому редкие, но разрекламированные успехи разносят вирус «золотой лихорадки» и поддерживают существование методик «героя-программиста».

Разработка после «лихорадки»

Разработка ПО в «постлихорадочный» период характеризуется более методичным, капиталоемким, менее рискованным и более трудоемким подходом. В проектах задействуются большие группы разработчиков, более формальные процессы, возрастает уважение к стандартам (к совместимости с предшествующим ПО, отраслевым протоколам и т. д.), расширяется программная база. Упор делается не на то, чтобы побыстрее вытолкнуть продукт на рынок, а на надежность, взаимодействие его с другими системами, практичность. Разработки ориентированы на использование проектирования, что вряд ли имело значение во времена «лихорадки», но приобрело большой вес по мере «взросления» технологии ПО.

У разработки ПО по принципу «золотой лихорадки» еще меньше шансов на успех в «постлихорадочный» период, чем во времена рискованного бума. На заре новой технологии ПО на рынке было несколько сложившихся игроков или законченных продуктов. Барьеры, преграждавшие выход

на рынок, были низкими, а первые продукты – небольшими, несовершенными, но все-таки успешными. Как и во времена золотой лихорадки в Калифорнии, чтобы застолбить участок в начале развития новой технологии, требовалось меньше людей и капитала. Первая версия редактора Word для Макинтоша была настоящим продуктом лихорадки – она состояла всего из 153 тысяч строк программного кода. Но по мере формирования зрелой технологии легко добываемое золото быстро заканчивается, и компаниям приходится конкурировать в более капиталоемких проектах.

Одна из самых серьезных ошибок успешных компаний, продолжающих работать по принципу «лихорадки», в том, что они продолжают следовать этим принципам, хотя уже сформировалась технология, а масштаб проектов неизмеримо вырос. Чтобы успешно конкурировать в постлихорадочный период, надо сделать значительно больше, чем просто увеличить персонал и найти гараж побольше.

Клиенты постлихорадочного периода более требовательны. Клиентов организаций-разработчиков ПО времен «золотой лихорадки» можно назвать «инноваторами» или «последователями-энтузиастами» и «экспертами», говоря словами Эверетта М. Роджерса (Everett M. Rogers), автора книги «Diffusion of Innovations» (Распространение инноваций) [119]. У таких людей есть вкус и пристрастие к новинкам технологий и умение прощать шероховатости, которые всегда их сопровождают. Продукты времен лихорадки, возможно, не столь блестящие, как в более поздние времена, но тем не менее могут быть успешными. Клиентов «постлихорадочного» периода Роджерс выделил в следующие группы: «раннее большинство», «позднее (зрелое) большинство» и «медлящие». Они отвергают риск и требуют тщательно проработанных продуктов, которые надежно работают. Это высоко поднимает планку для продуктов «постлихорадочного» периода. Программа сегодняшней версии редактора Word для ОС Windows содержит 5 миллионов строк.

Парадокс процессов «лихорадки» состоит в том, что компании, успешные во время одной «лихорадки», имеют шанс провалиться во время следующей. Подтверждением этому являются некоторые компании, организованные непосредственно во время лихорадки. Они повторяют ошибку Маршалла и Саттера, считая новые технологии досадной помехой их тщательно разработанным планам по получению максимальной отдачи от участков, которые они уже застолбили. Примерами компаний, опоздавших выхватить самородки новых технологий, могут служить IBM на заре развития MS DOS для ПК, Lotus в начале развития Windows и даже Microsoft, ко-

гда только забрезжил рассвет Интернета, хотя Microsoft быстро попыталась исправить эту ошибку. Но самым убедительным примером в современной компьютерной отрасли должен быть Xerox. Многие фундаментальные изобретения эры настольных компьютеров были сделаны в исследовательском центре Xerox в Пало-Альто, включая графический интерфейс пользователя, ручной манипулятор (мышь), стандарт локальной сети Ethernet. Однако компания была настолько поглощена ведением войны за свой бизнес копировальной техники, что проиграла войну в компьютерных технологиях, так по-настоящему и не вступив в нее.

Другие же компании «постлихорадочного» периода слишком неповоротливы, чтобы успешно конкурировать на рынках периода «новой волны». Крупные издержки, которые компании терпят при продаже продуктов клиентам «раннего» и «позднего большинства», не возникают во время лихорадки. На рынке в период лихорадки можно представить «голый» продукт, и он все равно пойдет хорошо у инноваторов и последователей-энтузиастов, преобладающих на таком рынке.

Несомненно, мы станем свидетелями повторов этой картины взрывного роста и резкого падения во время следующих циклов развития технологий, что бы ни последовало за Интернетом. Некоторые компании, добившиеся громадных успехов на начальной стадии развития Интернета, такие как Amazon.com, eBay и Yahoo, упустят следующую волну, и только время покажет, кто из них успешно преодолеет следующие великие перемены.

Смысл и бессмыслица экономики золотой лихорадки

С точки зрения макроэкономики тысячи добровольно рискующих индивидуальных разработчиков ПО, из которых обогатятся лишь несколько счастливчиков, а остальные спишут свои убытки на приобретение опыта, – это огромный актив. За свои неудачи расплачиваются они сами и никто другой, а все остальные только выигрывают от возможности приобрести и использовать лишь успешные продукты. Но как отдельная компания может извлечь выгоду из этой динамики? Какая компания может позволить себе финансировать тысячи индивидуальных предпринимателей во время очередной лихорадки, чтобы найти одного или двух, которые откроют «золото» новой технологии? Даже компании с широкими возможностями исследований, такие как AT&T, IBM, Microsoft, Xerox, не могут обеспечить финансирование тысяч проектов в каждой новой области технологии, и это одна из причин, по которой поглощение компаний ПО

во время лихорадки более разумно, чем это может показаться на первый взгляд. Некоторые аналитики отрасли сочли безумством приобретение компанией Microsoft за 130 миллионов долларов фирмы Vermeer Technology, изобретателя FrontPage, когда вся выручка Vermeer за год составляла 10 миллионов. Однако с точки зрения предпринимательского бума заплатить во время лихорадки 130 миллионов долларов за один успешный проект из тысячи – весьма недорогая альтернатива финансированию тысячи внутренних тупиковых экспериментальных разработок.

Расширение и сжатие

Методики инженерии ПО в «постлихорадочный» период безоговорочно доказали свою ценность в крупных проектах. (Сомневающиеся могут обратиться к главе 13.) Но у них есть резервы и для небольших проектов. Ларри Константин описывает соревнования Software Challenge (Вызов ПО) Австралийского компьютерного общества, во время которых группа из трех разработчиков должна была создать и запустить программу-приложение из 200 функциональных блоков за 6 часов [26]. Это непростая задача, эквивалентная написанию примерно 20 тысяч строк кода на традиционном языке третьего поколения или около 5 тысяч строк на языке визуального программирования.

Группа из компании «Ernst and Young» решила следовать формальной методике разработки – сжатой версии их обычной методологии с поэтапными наработками и промежуточными сдачами материала. Их подход включал тщательный анализ требований и проектирования. Многие из их соперников сразу взялись за написание программ, и первую пару часов команда «Ernst and Young» отставала.

Но к середине дистанции эта команда получила подавляющее преимущество. Однако она проиграла, но не потому что их систематический подход не удался. Они просто случайно стерли некоторые файлы, и их работа за полдня пошла насмарку, так что они смогли продемонстрировать меньшую функциональность к концу состязания, чем на середине дистанции.

Выиграла бы команда «Ernst and Young», не случись этого казуса с управлением конфигурацией? Ответ положительный. Эта группа через несколько месяцев снова выступила на соревновании на скорость разработки ПО, и на этот раз она позаботилась о создании резервных копий и контроле версий. И победа пришла [27]. Успех был достигнут не за счет обеднения предыдущего подхода, а за счет нахождения и устранения его слабых мест.

Ценность общего применения систематического совершенствования процессов в небольших организациях была подтверждена в исследовании, проведенном Институтом инженерии ПО [57]. Доля успешных программ совершенствования процессов в организациях с числом разработчиков менее 50 оказалась такой же, как и в крупных компаниях. Более того, в небольших организациях оказалось меньше проблем, препятствующих успеху (таких как организационная политика и защита узких полномочий и привилегий), чем в крупных фирмах.

Назад к «золотой лихорадке»

Проектам ПО времен лихорадки присущ изначальный риск, но неупорядоченность методики разработки ПО делает риск еще большим. Разработчики, действующие по проектам периода лихорадки, десятилетиями были вооружены программным эквивалентом ведер и лопат. В результате многие находки, идеи и инновации бездарно пропадали, точно так же, как из-за отсутствия контроля кода источника была утеряна часть программ команды «Ernst and Young».

Систематический подход к инженерии ПО необходим для успеха проектов периода после лихорадки, но он столь же полезен в проектах, все еще пребывающих в стадии лихорадки. Что случилось бы, если бы разработчики исходных версий VisiCalc, Lotus 1-2-3, NacOS, броузера Mosaic Web и других революционных проектов затерли свои рабочие файлы? О скольких инновационных продуктах мы так и не услышали бы, потому что разработчики стерли свои файлы? А сколько продуктов пало жертвой более серьезных ошибок?

Во время лихорадки можно быть чрезвычайно неаккуратным и не слишком изобретательным и все-таки сделать состояние, однако шансов на это немного. После лихорадки приходится быть более организованным и гибким, чтобы просто оставаться рентабельным. Предпринимательский зуд, которым заражается участник проекта во время лихорадки, – это, конечно, одно из самых увлекательных ощущений в жизни, однако никакого противоречия между предприимчивостью и применением эффективных методик разработки ПО нет. Изучая методы, которые хорошо зарекомендовали себя в периоды после лихорадки, можно глубже понять те из них, которые лучше всего сработают при наступлении очередной лихорадки, увеличив шансы на отыскание «золотой жилы».

ГЛАВА ТРИНАДЦАТАЯ

Необходимость совершенствования методик разработки ПО

Если можно измерить предмет обсуждения, выразить его числами, то об этом предмете что-то становится известно. Но если его невозможно измерить, нельзя найти числовое выражение, то знания о нем скудны и неудовлетворительны.

Лорд Кельвин, 1893 г.

Компании, вложившие средства в практические методики разработки ПО после лихорадки, получили отдачу от этих инвестиций. В 1944 г. Джеймс Хербслеб (James Herbsleb) сообщал, что средняя «прибыльность бизнеса» (в грубом приближении – рентабельность инвестиций) 13 организаций, проводивших программы систематического совершенствования практических методов, оказалась равной почти 500%, причем у лидеров она достигала 900% [56]. В 1999 г. Нил Ольсен (Neil C. Olsen) опубликовал аналогичные результаты для компаний, которые сделали серьезные вложения в подбор персонала, обучение и создание рабочей обстановки [102]. В 1997 г. Рини ван Солинген (Rini van Solingen) также приводил данные о рентабельности таких инвестиций – в среднем около 700%, причем в лучших организациях она достигала 1900% [136]. В 2000 г. Кейперс Джоунз отметил, что отдача от инвестиций в совершенствование процессов может легко достигать четырехзначных чисел, т. е. превышать 1000% [68]. Недавно проведенный Уоттсом Хамфри (Watts Humphrey) анализ рентабельности инвестиций в совершенствование методик разработки ПО показал, что она может достигать 500% и выше [60].

Состояние на практике

Большинство читателей, наверное, предполагает, что эффективность фирм, производящих ПО, следует типичной кривой нормального распределения: основная масса находится где-то посередине, а по краям совсем неэффективные организации и немного исключительно эффективных. Эта картина представлена на рис. 13.1.

В противоположность ожиданиям реальность совсем иная. В силу медленного принятия передовых эффективных методик, о чем говорилось в главах 1–2, лишь горстка организаций ПО работает по высшему классу. Аналитики отрасли уже давно подметили огромный разрыв между лучшими и худшими организациями, работающими в одной отрасли, определив их соотношение как 10:1 [91]. Средние организации значительно ближе к худшим, чем к лучшим. Такая реальная картина представлена на рис. 13.2.

Рис. 13.1. *Распространено мнение, что эффективность распределяется симметрично с примерно равным количеством результативных и нерезультативных предприятий ПО*

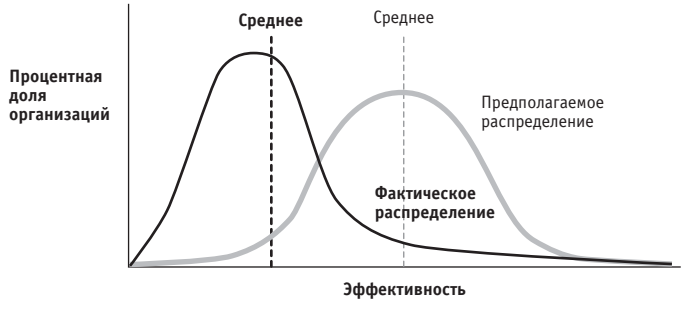

Рис. 13.2. *Реальное распределение эффективности фирм, производящих ПО, несимметрично. Большинство фирм ближе к худшим, чем к лучшим, в смысле практических методик [112]*

Выигрыш от совершенствования практических методик разработки ПО

Глубокое исследование 13 фирм-производителей ПО, проведенное Институтом инженерии ПО, показало, что типичная (средняя) фирма, систематически совершенствующая методики, имела рост производительности, равный 35% в год, сокращение сроков разработки составило 19% в год, снижение количества дефектов, обнаруженных после выпуска, – 39% в год. Эти числа создают основу для расчета рентабельности инвестиций. Результаты приведены в табл. 13.1.

Лучшие фирмы добились еще более впечатляющих результатов. Предприятие с наивысшим приростом производительности повышало ее на 58% ежегодно в течение 4 лет с совокупным ростом более 500%. Лучший показатель по сокращению сроков разработок составил 23% в год в течение 6 лет подряд с совокупным сокращением сроков на 91%. Лучшее долгосрочное повышение качества разработок выразилось в сокращении дефектов на стадии после выпуска на 39% в течение 9 лет с совокупным снижением числа обнаруженных дефектов на 99%. Две организации добились краткосрочного снижения числа дефектов на 70% и больше на протяжении чуть менее 2 лет.

Таблица 13.1. Результат работы по совершенствованию процесса разработки ПО

Показатель	Среднее улучшение	Наилучшее устойчивое улучшение
Производительность	35% в год	58% в год
Сокращение сроков	19% в год	23% в год
Снижение числа дефектов после выпуска	39% в год	39% в год*
Ценность для бизнеса организационного совершенствования	500%	880%

* «Средние» и «устойчивые» значения в данном случае одинаковы, поскольку лучшие результаты по сокращению числа дефектов были краткосрочными и не рассматривались как устойчивые.

Там, где господствует принцип «напишем и исправим», часто делают ставку на некую «золотую середину» между небольшим количеством дефектов и производительностью. Но, как говорилось в главе 2, большая часть затрат в таких проектах обусловлена именно незапланированными

исправлениями уже написанных программ. Данные табл. 13.1 подтверждают, что для большинства организаций не существует компромисса между более высокой производительностью и улучшением качества. Организации, заострившие внимание на уменьшении количества дефектов, в результате добиваются сокращения сроков и повышения производительности.

Доля организаций, систематически совершенствующих практические методы разработки ПО, относительно мала. В абсолютном выражении сотни организаций работают над этим; многие из них опубликовали результаты этой деятельности в профессиональных журналах, трудах конференций и других изданиях. В табл. 13.2 содержатся сводные данные о рентабельности инвестиций в эту работу по сообщениям 20 организаций.

Таблица 13.2. Примеры рентабельности инвестиций (ROI) в совершенствование разработки ПО [39], [74], [136]

Организация	Результат
BDN International	ROI 300%
Boeing Information Systems	Оценка в пределах 20%, за год сэкономлено 5,5 млн. долл., ROI–775%
Computer Sciences Corporation	Снижение на 63% показателя числа ошибок
General Dynamics Decision Systems	Сокращение переделок на 70%; снижение на 94% числа дефектов; рост производительности в 2,9 раза
Harris ISD DPL	Сокращение на 90% числа дефектов; рост производительности в 2,5 раза
Hewlett-Packard SESD	ROI – 900%
Hughes	Сокращение на 2 млн долл. в год перерасходов сметы, ROI – 500%
IBM Toronto	Сокращение дефектности при поставке на 90%, сокращение переделок на 80%
Motorola GED	Повышение производительности в 2–3 раза, сокращение цикла разработки в 2–7 раз, ROI – 677%
Philips	ROI – 750%
Raytheon	ROI – 770%
Schlumberger	Сокращение дефектности в 4 раза при бета-отладке
Siemens	Сокращение дефектности при выпуске на 90%
Telcordia	Дефектность 1/10 от среднеотраслевого уровня, рост удовлетворенности клиента с 60% до 91% за 4 года

Таблица 13.2 (продолжение)

Организация	Результат
Texas Instruments – Systems Group	Сокращение дефектности при поставке на 90%
Thomson CSF	ROI – 360%
BMC США	ROI – 410%
Центр авиалогистики ВВС США в Огдене	ROI – 1 900%
Центр авиалогистики ВВС США в Оклахоме	ROI – 635%
Военно-воздушная база ВВС США в Тинкере	ROI – 600%

Показатели ROI для отдельных методик

Организации с разной степенью подробности описывают, каким образом они добились внушительного повышения эффективности разработки ПО. Тем не менее можно выявить некоторые конкретные методики, доказавшие свою эффективность в целом. В табл. 13.3 приведены приблизительные показатели ROI для отдельных методик.

Таблица 133. Рентабельность инвестиций для отдельных практических методик разработки ПО [64]

Методика	ROI за год, %	ROI за 3 года, %
Формальная проверка программ	250	1 200
Формальная проверка проектирования	350	1 000
Долгосрочное технологическое планирование	100	1 000
Инструменты оценки затрат и качества	250	1 200
Измерения производительности	150	600
Оценка процессов	150	600
Обучение управленческого персонала	120	550
Обучение технического персонала	90	550

Что дает анализ бюджетирования ПО

Точность оценки сметных расходов организации – это хороший показатель, характеризующий управление и исполнение проектов. Анализ 26 тысяч проектов разработки бизнес-систем, проведенный фирмой Stan-

dish Group, показал, что в среднем перерасход запланированных средств на проект превышает 100% [62], [132]. Согласно этому анализу ошибочность сметных расходов по порядку величин соответствует результатам других исследований отрасли [66]. На рис. 13.3 показаны результаты одного исследования проектов ВВС США на разных уровнях методики разработки ПО. (Этот анализ основан на подходе SW-CMM, который более подробно обсуждается в главе 14.) Каждая точка ниже уровня 100% обозначает проект, который не превысил сметную стоимость. Каждая точка выше уровня 100% обозначает проект, вышедший за пределы сметы.

Как видно из рис. 13.3, организации, использующие наименее сложные методики (уровень 1 по SW-CMM), регулярно превышают сметную стоимость проектов – другими словами, они все время недооценивают затраты на проект. Более четко функционирующие организации (находящиеся на уровне 2 по SW-CMM) равномерно распределяют сметные ошибки между завышением и занижением, однако относительная ошибка смет все-таки, как правило, составляет 100% или больше. У организаций, прибегающих к самым развитым методикам (они находятся на уровне 3 по SW-CMM), перерасход и экономия смет встречаются одинаково часто, а точность сметной стоимости значительно улучшается.

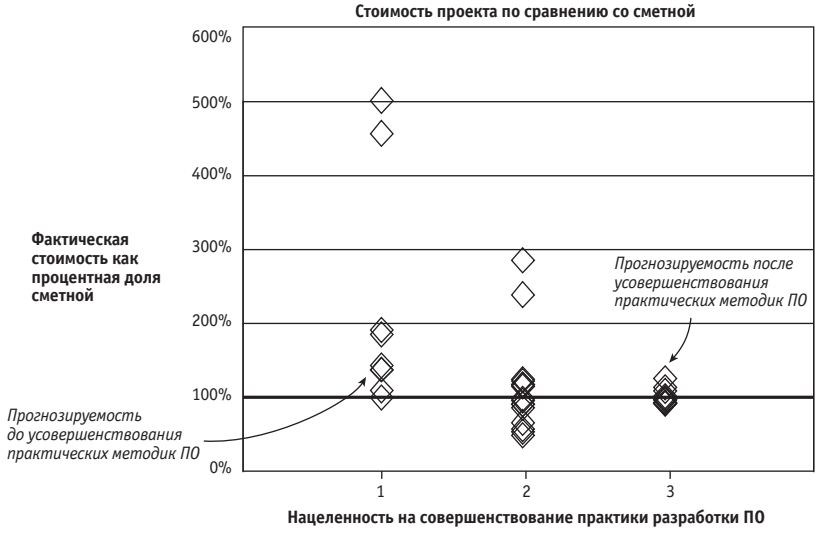

Рис. 13.3. *По мере совершенствования методик разработки ПО организации лучше контролируют сметные стоимости проектов, что в целом указывает на улучшение их контроля [78]*

Косвенный выигрыш от улучшения практических методик

Опубликованные показатели рентабельности инвестиций большей частью основаны на экономии операционных расходов, то есть на снижении стоимости написания одной строки программы или сданного функционального блока. Хотя эта экономия средств и впечатляет, еще большие выгоды компании могут получить в виде косвенной отдачи от улучшения практических методик. Более совершенные методики приводят к улучшению точности прогнозирования расходов и сроков, снижению рисков их превышения, выявлению проблем на ранних стадиях, а также содействуют лучшему управлению.

Сколько выиграет компания, разрабатывающая ПО, если ошибка определения срока выполнения проекта уменьшится с ±100 до ±10%? Как оценить выигрыш от возможности пообещать клиенту завершить проект на 6–12 месяцев раньше с высокой долей уверенности уложиться в эти сроки? Сколько стоит для компании, разрабатывающей единичные образцы продуктов для конкретного клиента, предложение выполнить проект за фиксированную цену и уверенность, что фактическая стоимость не будет значительно отличаться от предложенной? Какова цена планирования перехода на новую систему с идеальной точностью для организации розничной продажи? Какова стоимость уверенности, что переход состоится, как запланировано, 1 октября при минимальном риске захватить начало сезона праздничных покупок?

В отличие от операционных выгод, на которых сконцентрирована литература отрасли, непрямые выгоды совершенствования практики разработки ПО открывают дверь к дополнительным возможностям извлечения дохода. Для руководителей высшего звена они могут быть еще более привлекательными, чем простая экономия операционных расходов.

Взгляд на лучших

Дальнейшее изучение практики планирования проектов обещает соблазнительное улучшение процесса разработки ПО. В большинстве организаций с укрупнением проектов падает производительность разработчиков. В отличие от экономии за счет масштабов, наблюдаемой в других видах деятельности, при разработке ПО обычно действует обратное правило: увеличение масштабов ведет к *разбазариванию* средств.

В организациях, применяющих систематический подход к планированию, трудозатраты в проекте оцениваются по формулам, подобным следующей:[1]

$$\text{ТрЗат} = 2{,}94 \times \text{КСП}^{1{,}10}$$

Здесь ТрЗат – это трудозатраты в человеко-месяцах; КСП – килостроки программирования, то есть оценка объема программы в тысячах строк. Коэффициенты 2,94 и 1,10 получены эмпирически калибровкой данных уже выполненных организацией проектов. Эти величины применимы к типичной организации. Конкретное значение показателя степени (1,10) весьма существенно, поскольку это означает, что более крупные проекты требуют непропорционально больших трудозатрат по сравнению с небольшими проектами.

Лаборатория инженерии ПО НАСА (SEL) являет собой показательное исключение. Это первая организация, получившая награду IEEE за совершенство процесса разработки, и одна из самых изобретательных в сфере разработки ПО в мире. В SEL применяется следующая формула расчета трудозатрат [96]:

$$\text{ТрЗат} = 1{,}27 \times \text{КСП}^{0{,}986}$$

Несмотря на мелкий шрифт, показатель степени 0,986 указывает на громадную разницу между формулой трудозатрат в SEL и прочих организациях. Во всех других опубликованных моделях этот показатель превышает 1, а у SEL он меньше 1, что говорит о небольшой, но *экономии* на масштабах, которой удалось добиться. Привлекательность совершенствования процесса состоит в том, что достаточно зрелая организация может разрешить проблему крупных проектов – производительность отдельного члена группы разработчиков способна расти по мере укрупнения проектов. Хотя такие случаи очень редки, это логическое следствие специализации, которая обсуждалась в главе 10.

Суть проблемы – организационная

Многие организации переносят ответственность за совершенствование методик на низший уровень – уровень проекта. Рассматривая недавно факторы «умножения трудозатрат» в популярной модели оценки

[1] См., например, работу [14], где в качестве базового коэффициента приведен множитель 2,94 и как номинальный фигурирует показатель степени 1,10.

Cocomo II [14], я был поражен тем, каким небольшим их количеством располагает менеджер отдельного проекта. Из 22 факторов, используемых в Cocomo II для уточнения базовой оценки трудозатрат проекта, только три, по моему мнению, обычно контролируются куратором проекта (это уровень документации, архитектура и решение по рискам, а также возможность применения в других сферах). Многочисленные факторы диктует специфика бизнеса компании (сложность продукта, требования к надежности, изменчивость платформы, уникальность ПО и т. д.). Их трудно изменить, не меняя самого предприятия. Остальные не поддаются изменению индивидуальными проектами и должны отрабатываться самой организацией – это способности персонала, разработка ПО несколькими группами, преемственность персонала, зрелость рабочих процессов и др. По-видимому, именно в организационной сфере кроются самые мощные инструменты совершенствования практических методик разработки ПО.

Последний великий рубеж

В типичной ситуации, принимая решение о целесообразности инвестирования в бизнес, обычно сравнивают рентабельность вложений со стоимостью капитала. Инвестиции, дающие большую отдачу по сравнению со стоимостью капитала (с учетом всех факторов), будут правильными [43], [117]. (Это упрощенное толкование. Более подробное описание можно найти в источниках из списка цитированной литературы.)

Стоимость капитала обычно составляет около 10%. Во многих отраслях инвестиции с рентабельностью 15–20% будут считаться очень привлекательными. Однако усовершенствованные методики разработки ПО обещают не 15–20%, а намного больше. Согласно примерам из табл. 13.2 (а также исследованиям, упомянутым в начале этой главы), совершенствование практических методик обеспечивает рентабельность от 300 до 1900%, а в среднем 500%. Инвестиции с такой рентабельностью – это нечто экстраординарное, они практически беспрецедентны в бизнесе. Это выше, чем доходность акций интернет-компаний во время бума доткомов. Это выше, чем доходность любых спекуляций на товарных рынках. Это почти как выигрыш в лотерею. Ни с чем не сравнимая возможность для любого предприятия, еще не использующего такие методики.

Причина такой исключительно высокой рентабельности прямо связана с тем, что обсуждалось в главах 1 и 2, – отработанные методики известны десятки лет, но большинство организаций пренебрегают ими. Риск

от принятия этих методик ничтожен, отдача огромна. Все, что нужно, – это воля и решимость организации воспользоваться ими.

Десять трудных вопросов

Как писал лорд Кельвин более века назад, принимать решения, не имея количественных данных, очень трудно. Оказывается, многие организации не могут ответить на ключевые вопросы, связанные с их деятельностью в области ПО, например такие:

1. Каковы расходы на разработку ПО?
2. Какая доля проектов на данный момент укладывается в сроки и выделенный бюджет?
3. Каково среднее превышение сроков и сметы выполняемых вами проектов?
4. Какие из сегодняшних проектов, вероятнее всего, сразу же провалятся?
5. Какой процент расходов по проектам является результатом работ, которых можно было избежать?
6. Насколько удовлетворены (количественно) пользователи вашего ПО?
7. Какова квалификация вашего персонала по сравнению со средней в отрасли?
8. Насколько ваша организация по потенциалу сравнима с аналогичными отраслевыми учреждениями?
9. Насколько (количественно) ваша производительность повысилась за последние 12 месяцев?
10. Каковы ваши планы по повышению квалификации персонала и эффективности всей организации?

Организации, которые не могут ответить на эти вопросы, почти наверняка попадают в левую часть кривой на рис. 13.2. Во многих организациях никто никогда не задавал себе подобных вопросов и там лишь смутно осознают, что это нужно делать. Убедившись в целесообразности совершенствования практических методик разработки ПО, возможно, следует задать одиннадцатый вопрос: «Что мешает применению усовершенствованных методик разработки ПО теперь, когда предъявлены столь убедительные аргументы в пользу этого?»

ГЛАВА ЧЕТЫРНАДЦАТАЯ

Птолемеево мышление

Все модели неверны, но некоторые из них полезны.
Джордж Бокс (George Box)

Знание само по себе сила.
Френсис Бэкон

Несколько лет назад я выступал с лекцией на тему, которую озаглавил «История двух проектов». Ее целью было дать представление об инженерном подходе в разработке ПО. Я утверждал, что хорошо организованные компании часто преуспевают в сложных рискованных проектах, потому что применяют эффективные методики разработки ПО, а плохо организованные компании часто проваливаются даже в мелких проектах с невысоким риском, потому что применяют негодные методики. Обычно мое выступление принимали довольно доброжелательно, но однажды один из слушателей заявил: «Это прекрасный пример статичного линейного Птолемеева мышления, которое игнорирует динамику и сложность реальных проектов ПО».

Если вы еще не забыли школьный курс, Птолемей был астрономом, жившим около 100 года до нашей эры, который утверждал, что Солнце вращается вокруг Земли. Его теорию в 1543 г. сменили воззрения Коперника, считавшего, что Земля вращается вокруг Солнца. Таким образом, тот самый слушатель утверждал, что рекомендованный мной подход являл собою пример теории Птолемея, то есть что мои взгляды отстали как минимум на 400 лет!

Коперник отказался от теории Птолемея, когда обнаружил данные, не укладывавшиеся в нее. Точно так же данные о реальных проектах ПО с очевидностью говорят об эффективности нацеленных на инженерный под-

ход методик разработки ПО, который я отстаиваю, невзирая на упомянутое замечание слушателя.

Обзор подхода SW-CMM

Описываемая мной методика в каком-то смысле основана на модели зрелости для ПО, разработанной Институтом инженерии ПО (SEI). Впервые она была предложена в 1987 г. и на данный момент реализует самый известный и эффективный подход к систематическому совершенствованию организаций-разработчиков ПО. В ней указывается путь, по которому обычно проходят компании, чтобы достичь выгод и преимуществ, описанных в главе 13.

Организации-разработчики ПО классифицируются в рамках модели SW-CMM по пяти уровням:[1]

Уровень 1 – начальный. Процесс разработки ПО хаотичен. Проекты, как правило, реализуются с превышением сроков и бюджета. Организационными знаниями обладают лишь отдельные программисты. С их уходом исчезают и соответствующие знания. Успех в основном зависит от вклада индивидуальных «героев», типы которых описаны в главе 7. В таких фирмах процветает метод «напишем и исправим». Разработчики находятся на этом уровне по умолчанию, если целенаправленно не внедряют более эффективные подходы к разработке ПО.

Уровень 2 – повторяемый. Базовые методики практического управления проектами определяются по каждому отдельному проекту, а организация обеспечивает их соблюдение. Успех проектов уже не зависит от отдельных личностей. Сила организации, находящейся на этом уровне, заключается в повторяемости опыта, полученного в аналогичных проектах. У таких организаций возможен сбой при встрече с новыми инструментами, методиками или типами ПО.

Уровень 3 – сформированный. На этом уровне применяются стандартизованные технические и управленческие процессы во всей организации, а индивидуальные проекты подстраивают стандартные процессы под конкретные нужды. Специальная группа ответственна за работы по процессам ПО. Внедрена программа обучения, обеспечивающая не-

[1] Лучшее полное описание модели SW-CMM содержится в работе [23]. Другую документацию можно загрузить с сайта SEI по адресу *www.sei.cmu.edu*.

обходимые знания и квалификацию менеджеров и технического персонала для работы на данном уровне. Эти организации ушли далеко вперед от практики «напишем и исправим» и регулярно сдают ПО в срок, укладываясь при этом в смету.

Уровень 4 – управляемый. Результаты выполнения проектов предсказуемы с высокой точностью. Процесс разработки достаточно сформировался, чтобы выявлять причины отклонения и справляться с ними. Организация накапливает информацию по проектам в базе данных с целью оценки эффективности различных процессов. Во всех проектах соблюдаются общие для организации стандарты управления процессами, так что накапливаемые данные могут осмысленно анализироваться и сравниваться.

Уровень 5 – оптимизирующий. Вся организация нацелена на постоянное реальное выявление и распространение методов усовершенствования процессов. Процессы варьируются, результаты изменений оцениваются, и успешные изменения принимаются как новые стандарты. Основное внимание в части обеспечения качества сфокусировано на предупреждении дефектов посредством обнаружения и устранения коренных причин их возникновения.

Исходный принцип подхода SW-CMM можно в какой-то степени отнести к закону Конвея: Структура компьютерной программы отражает иерархию создавшей ее организации [30]. В хаотичных организациях рождается беспорядочное ПО. Организации, прибегающие к помощи программистов-«героев», дающие им почти полную свободу ради создания чудес программирования, выдают поочередно блестящее и небрежное ПО. Раздутые организации с неэффективными процессами разработки выдают неряшливое и неупорядоченное ПО. Ну а эффективные, оптимизирующие свои процессы организации предположительно создают точно выверенное, удовлетворяющее пользователя ПО.

Движение вверх

Отрасль ПО добилась прогресса на пути, определенном в SW-CMM. Как показывает рис. 14.1, в 1991 г. из 132 оцениваемых организаций лишь около 20% соответствовали уровням выше первого [112].

Из рис. 14.2 видно, что в 2002 г. из 1978 анализируемых организаций уже две трети функционировали на уровне выше первого.

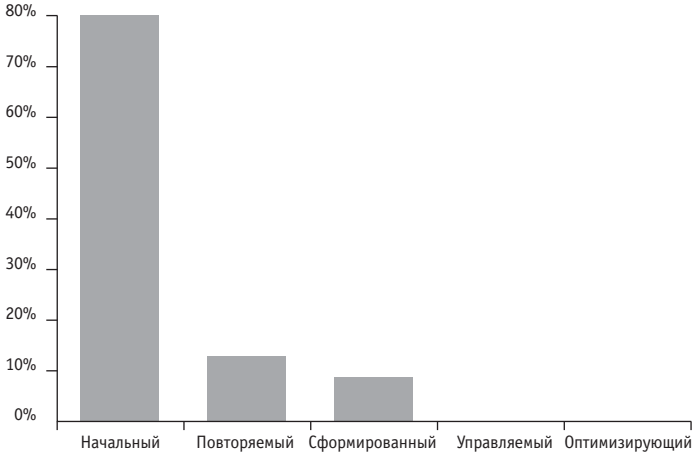

Рис. 14.1. *Профиль оценки организаций, проанализированных в соответствии с SW-CMM в 1991 г. Источник: «Профиль зрелости в отрасли ПО на конец 2002 г.»*

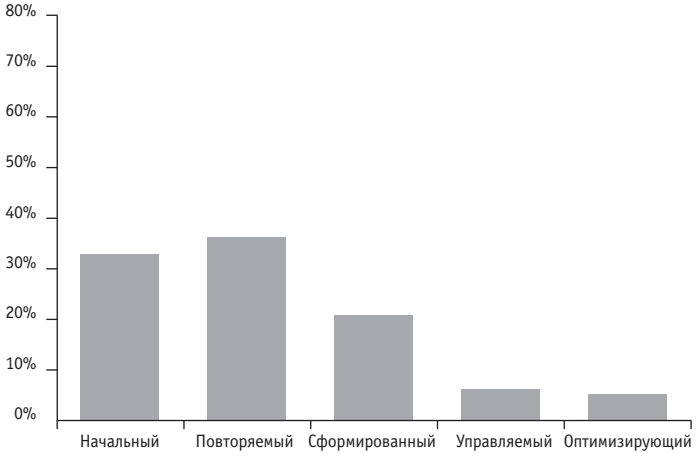

Рис. 14.2. *Профиль оценки зрелости организаций на конец 2002 г. Источник: «Профиль зрелости в отрасли ПО на конец 2002 г.»*

Тенденция, очевидная из анализа этих двух диаграмм, выглядит обнадеживающе, однако лишь малая доля организаций ПО Северной Америки была охвачена исследованием, и можно смело предположить, что общая картина скорее похожа на наблюдавшуюся в 1991 г., чем в 2001 г. Другими

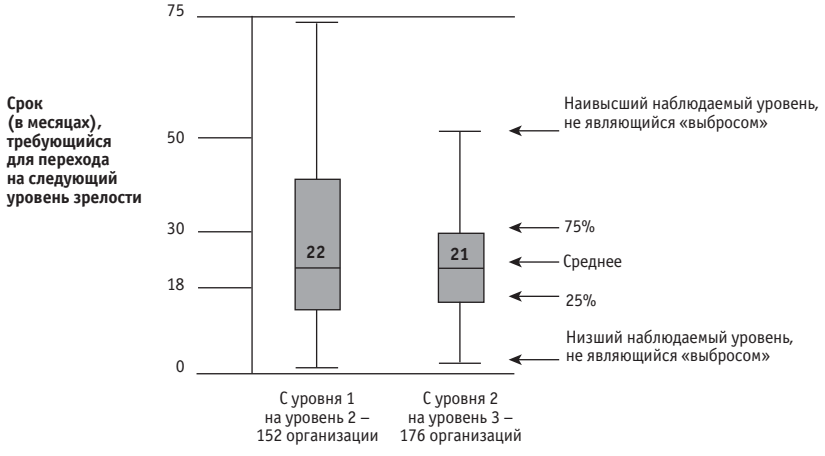

Рис. 14.3. *Перемещение организаций с одного уровня SW-CMM на следующий с 1992 г. Источник: «Профиль зрелости в отрасли ПО на конец 2002 г.»*

словами, около 75% организаций-разработчиков ПО функционируют на уровне 1.[1]

Может ли средняя организация добиться результатов в русле выявленной тенденции? Ответ: несомненно, да. Как видно из рис. 14.3, Институт SEI исследовал свыше 300 организаций, пытавшихся поднять свой рейтинг согласно SW-CMM от уровня 1 до уровня 2 или от уровня 2 до уровня 3 [112]. В течение трех с половиной лет или даже меньше 75% смогли выйти на уровень 2, при этом средний срок составил 22 месяца. За два с половиной года или менее 75% организаций смогли перейти на уровень 3 с уровня 2, а средний период составил 21 месяц. Организации, целенаправленно совершенствующие свою деятельность в русле SW-CMM, обычно добиваются серьезных успехов, причем достаточно быстро.

Все риски, с которыми можно справиться

Тот самый слушатель моей «Птолемеевой» беседы утверждал, что мой подход игнорировал динамику реальных проектов; были также и люди, утверждавшие, что SW-CMM культивирует склонность организаций отказываться от риска [37]. Организации становятся консервативными и за-

[1] Этот факт также подтверждает Кейперс Джоунз в [66].

бюрократизированными, что плохо влияет на их конкурентоспособность. Подтверждается ли это отраслевым опытом?

В организациях уровня 1 менее половины участников опроса заявили, что их руководство готово принять «умеренный» или «существенный» риск. В то же время в организациях уровня 3 почти 80% респондентов заявили о готовности принять такую степень риска [57].

Эта связь между эффективностью процессов компании и ее способностью принимать риски была продемонстрирована в проекте ATAMS в Шайенских горах [49], [114], [131]. По этому проекту группа разработчиков взялась выполнить некую разработку за одну пятую стоимости и за половину срока лучшего из имевшихся предложений. В конце концов ПО было сдано за месяц до окончания срока без перерасхода выделенных средств. За 18 месяцев прошедших с выпуска в ПО было обнаружено только 2 дефекта, которые были легко устранены. Успех группы разработчиков в этом достаточно рискованном предприятии объяснялся тщательным контролем требований, применением формальных экспертиз как программ, так и общей структуры, активным контролем риска.

Влияние организационных усовершенствований на готовность принять риск противоположно тому, что предсказывают некоторые специалисты. Фирмы, в которых применяются проработанные процессы, снижают воздействие необязательных рисков (Фредерик Брукс назвал бы их «случайными», в противовес «существенным»), гораздо лучше приспособлены для принятия взвешенных добровольных рисков, чем их менее подготовленные собратья, которых со всех сторон захлестывают вынужденные риски. (В главе 5 различия между случайными и существенными качествами по Ф. Бруксу обсуждаются подробнее.)

Кто применяет SW-CMM?

С 1987 г. были оценены возможности примерно 2000 организаций, результаты свыше 10 000 проектов были направлены в Институт SEI. Три четверти организаций, участвующих в повышении уровня согласно SW-CMM, – это коммерческие фирмы-разработчики или компактные группы разработчиков, представляющие различные отрасли: финансы, страхование, недвижимость, оптовые продажи, строительство, транспорт, связь, ЖКХ, машиностроение, электронное оборудование, медицинские инструменты и множество других. Почти четверть всех фирм выполняли разработки по государственным заказам. Около 5% представляли оборонные

или правительственные агентства. Размеры этих организаций менялись в широких пределах. Почти половина имела в штате менее 100 разработчиков. Персонал четверти организаций превышал 200 человек, и примерно столько же организаций насчитывали менее 50 разработчиков.

Бездушная разработка ПО

Одно из самых распространенных возражений против организационного совершенствования связано с тем, что при этом насаждается бюрократия, ограничивающая творчество. Это напоминает древнее возражение, что искусство и инженерия несовместимы. Конечно, можно создать гнетущую обстановку, когда творчество и задачи бизнеса расходятся; точно так же можно построить и ужасные здания. Но столь же возможно создание гармонии между запросами разработчика ПО и задачами бизнеса, что подтверждается отраслевыми данными. Среди принявших участие в опросе о влиянии SW-CMM 84% категорически не согласились с утверждением, что совершенствование в SW-CMM сделало их организации более жесткими или бюрократическими.

Организации, нацеленные на организационное совершенствование, обнаружили, что эффективность практических процессов стимулирует творчество и высокий моральный дух работников. В опросе 50 организаций только 20% сотрудников компаний уровня 1 охарактеризовали моральный дух персонала как «высокий» или «отличный» [57]. Приведенные ответы коррелировали по группам менеджеров и разработчиков, отвечающих за организационные усовершенствования, и старшего технического персонала в целом. В организациях, которым был присвоен уровень 2, доля сотрудников, оценивающих моральный дух своих коллег как «высокий» или «отличный», подскочила до 50%. В организациях уровня 3 уже 60% опрошенных оценили свой моральный дух таким образом.

Эти сводные статистические данные подтверждаются более глубоким анализом организаций, получивших высшие оценки эффективности процессов разработки. Опрос, проведенный в Центре авиалогистики в г. Огдене – одной из первых организаций, получивших рейтинг уровня 5 по SW-CMM, – показал, что сотрудники сферы ПО были воодушевлены изменениями, которые явились результатом восьмилетних усилий по совершенствованию организационных процессов [101]. Респонденты действительно считали, что процессы уровня 5 сужают выбор методов выполнения работы, но ограничения рассматривались как неизбежный сопутствующий эле-

мент повышения эффективности и не считались отрицательным фактором. Сотрудники сферы ПО сочли, что выполнение работ значительно облегчило внедрение организационных усовершенствований. Подавляющее большинство сотрудников признало, что теперь они вносят больший вклад в планирование и контроль проектов. Каждый респондент утверждал, что инициатива совершенствования SW-CMM оказала положительное влияние.

Группа разработчиков бортового ПО для космических кораблей «Шаттл» НАСА в Центре космических полетов имени Джонсона – еще одна организация, которая получила уровень 5 по SW-CMM [44]. Там нет гор коробок из-под пиццы, пирамид банок из-под кока-колы, отвесных стен для альпинистов, площадок для катания на роликовых досках или подобных элементов пейзажа «продвинутых» организаций-разработчиков ПО. Там не в игрушки играют, а стремятся создавать безукоризненное ПО. Работа интересная, но не всепоглощающая. Группа разработчиков ПО работает обычно 5 дней в неделю по 8 часов. В отрасли с выраженным преобладанием мужского персонала почти половина группы разработчиков женщины.

Люди, которые уходят из таких групп с высокой производительностью, бывают поражены неэффективностью средних организаций. Один из таких сотрудников сменил группу в Центре космических полетов имени Джонсона на более предприимчивую среду. Через несколько месяцев он вернулся. Оказалось, что организация, в которую он уходил, поддерживала эффективность в разработке ПО лишь на словах, а на деле исповедовала тот самый принцип «напишем и исправим». В обстановке высокого уровня зрелости сотрудники не только не считают его каким-то образом ограничивающим их творчество, но достигают уровней производительности и качества, просто невозможных в менее зрелых компаниях. В упоминавшемся выше проекте ATAMS применялись высоко структурированные методики, способные показаться стесняющими некоторым программистам, однако участники группы ATAMS говорили, что структурированный процесс позволил реализовать высшую производительность каждого участника. По их словам, они с неохотой взялись бы за разработку ПО, если бы этот процесс не применялся.

Серьезная самоотдача

Одна из не слишком привлекательных особенностей организационного совершенствования состоит в том, что его нелегко добиться. Джеймс Хербслеб провел опрос руководителей, которые внедрили усовер-

шенствование по SW-CMM. 77% опрошенных заявили, что этот процесс оказался более продолжительным, чем ожидалось, а 68% сказали, что он стоил дороже, чем предполагалось [57].

Успех совершенствования в SW-CMM зависит от следующих факторов:

- Самоотдача менеджеров высшего звена, которые должны обеспечить руководство и финансирование, присвоение высшего приоритета долгосрочным усовершенствованиям, активный мониторинг динамики совершенствования процесса разработки ПО.
- Образование группы процесса инженерии ПО (SEPG). В крупной организации может потребоваться не одна такая группа. В нее должны войти должностные лица высокого уровня, осознающие цели совершенствования процессов в организации, связанные с этим вопросы культуры и свою роль как внутренних консультантов.
- Необходимая подготовка и обучение руководителей среднего звена и технического персонала наряду с вознаграждениями за повышение производительности, ориентированные на достижение долгосрочных целей SW-CMM.

Этот список весьма упрощен, и в каждой отдельной организации найдется еще ряд особых факторов, влияющих на ее инициативу организационного совершенствования. Как упоминалось в главе 2, некоторые организации внедряют SW-CMM, относясь к этому как к модному словечку. Попытки относиться к SW-CMM как к очередной панацее вряд ли будут успешны.

Рейтинг организаций

SW-CMM – это сложившаяся эффективная модель организационного совершенствования. Структурируя организации по пяти уровням, мы получаем также эффективный способ их оценки. В других, более устоявшихся профессиях оценка организаций представляет собой обычную часть программы поддержания высоких практических стандартов. Бухгалтерские фирмы каждые три года должны проходить аттестацию. Аттестация колледжей действительна максимум три года, после чего она пересматривается. Некоторые индивидуальные программы колледжей аттестуются отдельно. Больницы проходят аттестацию в Объединенной комиссии по аттестации организаций здравоохранения (Joint Commission on Accreditation of Healthcare Organizations – JCAHO) и получают ее максимум на три года.

JCAHO указывает следующие причины, по которым больницы стремятся пройти аттестацию:[1]

- это повышает доверие пациентов;
- дает возможность отчитываться по специальной форме перед общественностью;
- обеспечивает возможность объективной оценки работы организации;
- стимулирует усилия по повышению качества работы организации;
- способствует привлечению профессионального персонала;
- служит образовательным средством для персонала;
- может использоваться для удовлетворения некоторых требований программы страховой медицины Medicare;
- ускоряет расчеты с третьими сторонами;
- часто обеспечивает соблюдение лицензионных требований штатов;
- может благоприятно повлиять на размер премий по страховым возмещениям;
- положительно влияет на принятие решений по договорам контролируемого медицинского обслуживания.

Аттестация больниц – дело добровольное. Однако большинство больниц стараются ее получить в силу вышеназванных причин.

Параллель с организациями ПО очевидна. Оценка по SW-CMM обеспечивает форму отчета, которую потенциальные клиенты могут использовать при принятии решения об условиях контракта на разработку ПО или приобретения его пакета. Это объективный общепризнанный стандарт для сравнения. Этим стимулируются усилия организаций по повышению качества, поощряя их повышать свой рейтинг. Можно предположить, что страховые компании предложат более выгодные ставки страховых полисов компаниям, имеющим достаточно высокий уровень зрелости.

Форма и содержание

Старая поговорка утверждает, что успех равен планированию, помноженному на исполнение.

Если планирование и исполнение выразить в процентах, то получится величина успеха от 0 до 100%. Если один из компонентов – планирование или исполнение – отсутствует, то успех будет равен нулю.

---

[1] См. сайт JCAHO по адресу *www.jcaho.org*.

По мере достижения отраслью ПО более высокого уровня профессионализма приходится учитывать все больший объем опыта, накопленного в связи с SW-CMM, как положительного, так и отрицательного.

В SW-CMM важно содержание, а не форма. Организации, стремящиеся к совершенствованию по SW-CMM исключительно ради присвоения уровня 2 или 3, вполне вероятно, ограничатся полусырым планированием и не слишком тщательным исполнением. Вряд ли им удастся получить желаемые рейтинги и добиться выигрыша в качестве и производительности, к которым они стремятся. Вот это и есть настоящее Птолемеево мышление, когда содержание вращается вокруг формы, вместо того чтобы находиться в центре солнечной системы ПО.

Те же организации, что нацелены на выигрыш от конечного качества и производительности в рамках совершенствования по модели SW-CMM, вероятно, более серьезно отнесутся к планированию и будут лучше выполнять планы. Внимание к процессу разработки позволяет организациям повышать производительность, создавать ПО с меньшим количеством дефектов, принимать больший риск, повышать точность прогнозирования, укреплять моральный дух и улучшать выполнение крупных проектов. Если посмотреть на уровень эффективности, которого достигают организации с высокой степенью отлаженности технологических процессов, становится ясно, что устаревшие практические методики вроде разработки по принципу «напишем и исправим» и есть настоящее Птолемеево мышление.

ГЛАВА ПЯТНАДЦАТАЯ

Количественное выражение факторов, связанных с персоналом

Качества персонала и работа с людьми – это, без сомнения, самый крупный источник возможностей повышения производительности разработки ПО.

Барри У. Боэм

Часто при обсуждении лучших практических методик, моделей совершенствования процессов и других достаточно сложных аспектов проблемы забывают о роли, которую играют людские факторы в эффективности разработки ПО.

Факторы персонала

Одним из наиболее воспроизводимых результатов исследований в области инженерии ПО является существенная разница в производительности между отдельными разработчиками. Сакман, Эриксон и Грант обнаружили, что время отладки одной и той же программы различными разработчиками может отличаться до 20 раз [121]. Сравнения выполнялись в группе программистов, имевших не менее 7 лет опыта профессиональной работы.

Этот основной факт – минимум десятикратная разница в производительности – был затем многократно подтвержден, но мне думается, что даже в нем истинная разница в производительности практикующих программистов занижена. Том Демарко и Тимоти Листер провели соревнование по программированию, где 166 программистам было предложено выполнить одно и то же задание [36]. Оказалось, что производительность различных программистов даже в небольшом проекте отличалась в пять раз.

Билл Кертис (Bill Curtis) провел исследование с аналогичными результатами, в котором группе из 60 профессиональных программистов была поставлена, по его словам, простая отладочная задача [33]. Среди испытуемых, выполнивших задания, производительность различалась на порядок.

Модель оценки Cocomo II дает дальнейшие подтверждения значительных вариаций способностей персонала [14]. В этой модели используются различные факторы для корректировки оценки базовой трудоемкости. Влияние каждого фактора анализировалось с помощью обширной базы данных проекта Cocomo II. Семь из 22 факторов в модели связаны с персоналом.

Согласно этой модели коэффициент воздействия фактора опыта группы разработчиков в области разработки приложений на стоимость и трудоемкость проекта равен 1,51. В терминах модели Cocomo II это означает, что группе разработчиков с наименьшим опытом (последние 15%) в данной предметной области потребуется в 1,51 раз больше трудозатрат для выполнения проекта, чем самой опытной группе (верхние 10%) при прочих равных условиях. Опыт работы с технической платформой имеет коэффициент влияния 1,40, а опыт работы с языком программирования и инструментарием – 1,43.

Способности аналитиков (это относится к их профессионализму, а не к опыту) оценены коэффициентом 2,00. Способностям программиста был присвоен коэффициент влияния 1,76. Коэффициент влияния факторов, связанных с общением (размещение персонала и поддержка контактов между ними, таких как электронная почта, локальная сеть и т. д.) был принят равным 1,53. Влияние преемственности персонала получило 1,51 балла.

Влияние этих факторов сведено в табл. 15.1.

Отдельно взятые факторы «старшинства» (опыт в приложениях, в языке и платформах) имеют совокупный коэффициент влияния 3,02. А семь связанных с персоналом факторов в совокупности получают ошеломляющие 24,6 балла! Этим простым фактом во многом объясняется лучшая по отрасли производительность в Microsoft, Amazon.com и у других лидеров разработок, не уделяющих на первый взгляд должного внимания отлаженности процессов.

Таблица 15.1. Влияние персонала по модели Cocomo II

Фактор влияния по модели	Наименование в модели	Коэффициент влияния
Опыт в приложениях	APEX	1,51
Факторы общения	SITE	1,53
Опыт в языке программирования и инструментарии	LTEX	1,43
Преемственность персонала	PCON	1,51
Опыт в платформе	PLEX	1,40
Способности программистов	PCAP	1,76
Способности аналитиков	ACAP	2,00
Совокупный		**24,6**

Слабосильные программисты

Вариации эффективности говорят о том, что одни программисты намного более производительны, чем другие. Естественно, хотелось бы найти способ привлечь и удержать лучших. Но это только полдела. В исследовании Демарко и Листера [36] 13 из 166 испытуемых программистов *вообще не смогли* выполнить задание, а это почти 10% выборки. В исследовании Кертиса [33] 6 из 60 профессиональных программистов не справились с «простым» отладочным заданием – те же 10%.

Что же значит в реальном мире работа программистов, которые не могут справиться со своей работой? В упомянутых исследованиях результаты программистов, не справившихся с заданием, не учитывались. В реальном проекте не учитывать результат вряд ли получится. Поэтому программистам, не справляющимся со своей работой, потребуется либо много дополнительного времени, либо помощь других программистов. В реальном проекте эта десятикратная разница в производительности вполне может превратиться в *противостояние производительности и антипроизводительности*, ведь в конце концов кому-то приходится переделывать работу программистов, которые не справляются с заданием. Другими словами, присутствие в проекте программистов с низшей производительностью фактически отодвигает работу назад.

Однако низкая производительность как таковая – это не единственная проблема. Работая на пределе своих возможностей по написанию про-

грамм, программисты с низкой производительностью либо не способны, либо не желают соблюдать соглашения, принятые для программы, или технологические стандарты. Они не устраняют большинство дефектов или все таковые из своего блока программы до его интеграции с блоками, написанными другими программистами, или пока эти дефекты не затронут работу других людей. Они не могут достоверно оценить свою работу, потому что не знают, закончат ли они ее. С учетом нулевого вклада в проект и создания дополнительной работы для остальной группы не будет преувеличением назвать их «программистами с отрицательной производительностью». Данные исследований говорят о том, что примерно 10% профессиональных программистов, возможно, относятся к этой категории. Так что случайным образом отобранная группа из 7 программистов с вероятностью S будет иметь в своем составе одного программиста с отрицательной производительностью.

Физические условия

Еще одним результатом соревнования в исследовании Демарко и Листера является тот факт, что 25% лучших по показателям программистов располагали более просторными тихими помещениями, где их реже тревожили и отвлекали от работы телефонными звонками по сравнению с остальными 75%. Разница в площади помещений была не так заметна – 25% лучших программистов в среднем располагали 76 кв. футами площади (примерно 7,5 кв. м), а остальные 75% – площадью 46 кв. футов (4,54 кв. м).

Разница в производительности более существенна. Производительность разработчиков из лучшей четверти участников оказалась в 2,6 раза выше, чем у худшей. В терминах модели Cocomo II разница влияния обстановки в офисном помещении между лучшими (первые 25%) и худшими (последние 25%) равна 2,6.

Мотивация

Мотивацию обычно считают самым сильным фактором, влияющим на работу, и большинство исследований показывает, что его влияние сильнее, чем любого другого фактора [11].

Чтобы ни говорили критики о Microsoft, все согласятся, что эта фирма чрезвычайно преуспела в мотивации своих разработчиков. В изобилии можно услышать истории о 12-, 14- и даже 18-часовом рабочем дне, о про-

граммистах, неделями остающихся на месте. Я знаю разработчика, у которого была складная кровать, изготовленная на заказ, чтобы умещаться в его офисе. Microsoft называют в своем роде «бархатной потогонкой», что заставляет предположить среди прочего, что фирма, возможно, более чем успешно мотивирует своих сотрудников.

Подход Microsoft к достижению столь высокого уровня мотивации прост: он направлен непосредственно на моральное состояние работника. Каждой группе в Microsoft выделяются средства, которые группа может потратить на что угодно. Одни коллективы приобретают автоматы для воздушной кукурузы, вроде стоящих в кинотеатрах, другие выезжают на горнолыжные курорты или ходят в боулинг, а некоторые организуют пикники. Есть и такие, которые делают футболки или же снимают полностью кинотеатр для просмотра любимого фильма.

Microsoft также весьма активно использует неденежные вознаграждения. За год, проведенный в этой фирме, я получил 3 футболки, толстовку, пляжное полотенце и коврик для мыши. Я также поучаствовал в групповой поездке на поезде и замечательно пообедал в местном ресторане и еще в одном столь же приятном месте. Если бы я был штатным сотрудником, у меня было бы еще больше футболок, а также часы Microsoft, бирка участника проекта и огромная пластиковая награда «За сдачу» при завершении проекта. В денежном выражении все это стоит лишь несколько сотен долларов, но психологически гораздо больше.

Не пренебрегает Microsoft и личной жизнью своих разработчиков. Пока я работал там, я наблюдал, как к разработчику в соседней комнате каждый день приходила после школы его десятилетняя дочь. Она тихо выполняла домашнее задание в уголке, пока он работал. Никто в компании и ухом не вел.

Помимо прямого поддержания морального духа Microsoft с готовностью использует другие факторы, направленные на его сохранение, причем иногда делает то, на что в другие компаниях вряд ли решились бы. Мне случалось наблюдать, как методологической чистотой, дисциплиной программирования, контролем за спецификациями продукта и сроками завершения, доступностью руководства – почти всем – жертвовали ради морального духа. Мотивационная эффективность такого подхода, не учитывая и другие его достижения, говорит сама за себя.

Опытность персонала

Делая акцент на опыте сотрудников в модели Cocomo, многие ведущие организации признают важность старшего персонала. Много лет назад директор по развитию в Microsoft заметил, что он выделяет старший персонал как критический фактор успеха. Он сказал, что один из двух ключей к успеху такого продукта, как Microsoft Excel, заключался в том, что два опытных старших сотрудника, работавших над предыдущей версией продукта, продолжили работать в новом проекте.

В исследовании вышедших из-под контроля проектов, проведенном в Великобритании, менеджеры назвали «нехватку опытного персонала» одной из причин трудностей проектов примерно в 40% случаев значительного превышения сроков или бюджета [24].

Даже фирмы с сильной ориентацией на процессы разработки ПО признают важность человеческого фактора. Лаборатория инженерии ПО НАСА стала первой организацией, получившей приз компьютерного общества IEEE за достижения в развитии процессов разработки ПО. В третьем пересмотренном издании «Recommended Approach to Software Development, Revision 3» (Рекомендуемого подхода к разработке ПО) [95] в числе девяти первых рекомендаций указана такая: «начинайте проект с небольшим коллективом опытного персонала».

Что в итоге

Оказывается, что внимание к технической подготовке, преемственности персонала, опыту в предметной области бизнеса, удобному помещению, мотивации и другим факторам, ориентированным на человека, – это трезвый экономический расчет. Отрасль ПО нуждается в эффективном подходе к профессионализму, призванном выявить лучших по производительности, отсеять худших и поднять средний уровень программистов до высших показателей.

ГЛАВА ШЕСТНАДЦАТАЯ

Программа профессионального развития фирмы Construx[1]

Как компания может поддержать формирование настоящих профессионалов в области ПО? В главе 6 мы уже говорили, что общеотраслевая поддержка развития карьеры профессионалов ПО постепенно складывается, но она еще слаба. Лишь несколько университетов предлагают начальное образование в инженерии ПО, а выпускников по этой специальности намного меньше, чем требуется отрасли. Ситуация быстро улучшается, но все же пройдут годы, пока количество выпускников университетов станет достаточным.

Частные компании тоже не расположены поддерживать эффективное развитие карьеры программистов. Вместо *продвижения* по карьерной лестнице большинство работников отрасли ПО просто переходят от одного проекта к другому без какого-либо улучшения своих навыков. Совсем мало компаний ИТ делают слабые попытки предложить карьерный рост своим сотрудникам, а отрасль в целом не располагает ничем, что хотя бы отдаленно напоминало профессиональный рост, такой как у врача, адвоката, юриста или бухгалтера.

Профессионализм в ПО давно представляет для меня интерес, и несколько лет назад я взялся за формирование четкого порядка карьерного продвижения и поддержку профессионального роста инженеров ПО в моей компании Construx Software. Были определены конкретные цели программы развития наших инженеров ПО:

---

[1] Эта глава адаптирована из Белой книги фирмы Construx Software, озаглавленной «Construx's Professional Development Program», © 2002 Construx Software Builders, Inc. Используется с разрешения. Эту Белую книгу можно загрузить со страницы *www.construx.com/profession*. Я признателен Дженни Стюарт (Jenny Stuart), составившей оригинал Белой книги.

- *Расширение профессиональных навыков.* Повышение мастерства и квалификации наших сотрудников.
- *Планирование карьеры.* Создание последовательного плана повышения квалификации инженеров ПО во время работы в Construx.
- *Поддержка обычных специальностей ПО.* Поддержка всего контингента должностей, связанных с ПО, включая разработчиков, инженеров, тестеров, бизнес-аналитиков, менеджеров проектов, архитекторов ПО и работников других отраслевых специальностей.
- *Согласованность.* Наличие согласованных средств оценки работы персонала и повышения в должности независимо от технической специализации.
- *Обобщения для других компаний.* Возможность предлагать программы другим компаниям, чтобы поддержать профессиональный рост их специалистов в области ПО.

Области знаний в Construx

Стержень нашей программы профессионального развития – «лестница профессионального развития» (Professional Development Ladder, PDL), основанная на выделенных в SWEBOK областях знаний, описанных в главе 5. В Construx эти области знаний получили название «области знаний Construx» (Construx Knowledge Areas, CKA). Они определяют объем знаний, которым должен владеть и который должен уметь применять технический персонал. Как описано в главе 5, есть десять областей знаний:

- Требования к ПО
- Проектирование ПО
- Создание ПО
- Тестирование ПО
- Обслуживание ПО
- Управление конфигурацией ПО
- Качество ПО
- Управление инженерией ПО
- Инструментарий и методики инженерии ПО
- Инженерный процесс разработки ПО

Каждая из этих областей знаний в SWEBOK определена до нюансов, но мы пришли к выводу, что необходимы также конкретные интерпретации для наших условий (табл. 16.1).

Таблица 16.1. Описание областей знаний Construx (СКА)

Область знаний (СКА)	Описание
Требования	Идентификация, анализ, моделирование и документирование функций, реализуемых в ПО.
Проектирование	Определение структуры и динамического состояния системы на различных уровнях абстракции и посредством различных представлений системы.
Испытания/ тестирование	Работы, связанные с прогонами ПО, призванные выявить дефекты и оценить функциональные возможности.
Обслуживание	Работы, связанные с установкой, внедрением, переводом и эксплуатацией системы.
Управление конфигурацией	Организация и хранение артефактов системы, контроль и управление изменениями этих артефактов, определение вида сдачи системы заказчику.
Качество	Работы, выполняемые на статичных артефактах, связанные с обеспечением требований качества данного элемента ПО в настоящий момент или в будущем.
Управление инженерией	Все аспекты управления – от управления бизнесом и персоналом до управления проектами.
Инструменты и методики инженерии	Применение инструментария, технологий, методологий и способов инженерии ПО.
Процесс	Работы, связанные с измерением и повышением качества разработки ПО, соблюдением сроков, эффективностью, производительностью и другими характеристиками проекта и продукта.

Уровни способностей

Области знаний Construx обеспечивают упорядоченный способ организации знаний инженерии ПО, но этого недостаточно для определения способностей инженера ПО. Поэтому внутри каждой области компания Construx выделяет 4 уровня *способностей*: вводный, продвинутый, ведущий и мастерский. Эти 4 уровня обозначают продвижение в приобретении знаний и опыта в каждой области знаний. В каждой области описаны типы деятельности (чтение, групповые семинары и опыт работы), необходимой для достижения определенного уровня способностей. Эти четыре уровня приведены в табл. 16.2.

Таблица 16.2. Сводка уровней способностей

Уровень способностей	Описание
Вводный	Основные работы в данной области сотрудник выполняет под контролем и предпринимает серьезные шаги по развитию своих знаний и мастерства.
Продвинутый	Сотрудник эффективно и независимо выполняет работы в данной области, является примером для менее квалифицированных сотрудников, иногда выступает в роли наставника.
Ведущий	Сотрудник выполняет образцовые работы в данной области. Регулярно выступает наставником и ведущим в проекте и, возможно, на уровне компании. Рассматривается как крупный ресурс в данной области знаний.
Мастерский	Является эталоном работ в данной области и имеет богатый опыт участия во многих проектах. Ведет занятия и семинары, является автором брошюр или книг, которые расширяют объем знаний в данной области. Является лидером на уровне отрасли, признан как эксперт в данной области.

Было установлено, что способности зависят от сочетания опыта и знаний. Невозможно достигнуть ведущего уровня знаний инженерной дисциплины, если последние не базируются на опыте. Но опыт такого же уровня невозможен без владения самыми передовыми знаниями. Поэтому при расхождении между опытом и знаниями сотрудника общий уровень способностей обычно ближе к более низкому. Эта мысль проиллюстрирована на рис. 16.1.

		Опыт			
		Вводный	Продвинутый	Ведущий	Мастерский
Знания	Вводный	Вводный	Продвинутый	Продвинутый	–
	Продвинутый	Продвинутый	Продвинутый	Продвинутый	–
	Ведущий	Продвинутый	Продвинутый	Ведущий	Мастерский
	Мастерский	–	–	Мастерский	Мастерский

Рис. 16.1. *Общие способности как функциональное проявление знаний и опыта*

Ступени лестницы профессионального развития

Сочетание областей знаний и уровня способностей позволяет сформировать ступени «лестницы» профессионального роста. Ступени обеспечивают повышение профессионализма и продвижение по службе. Восхождение по лестнице требует от инженера как расширения знаний (охват большего числа областей знаний), так и их глубины (повышение уровня знаний в данных областях). Нужно приобретать и знания, и опыт.

По причинам исторического характера ступени лестницы имеют номера от 9 до 15. Выпускники колледжей обычно начинают со ступени 9, а опытные инженеры могут начать с 10-й или 11-й. Двенадцатая ступень считается в Construx полноценным профессиональным статусом. Многие инженеры решают не двигаться выше двенадцатой ступени, поскольку ступени с 13-й по 15-ю можно достичь, только сделав существенный инновационный вклад как в Construx, так и в инженерию ПО.

В табл. 16.3 перечислены все ступени лестницы и описаны требования для достижения каждой из них.

Таблица 16.3. Требования к освоению каждой ступени лестницы профессионального развития

Ступень лестницы	Описание	Охваченные области знаний (СКА)
9	Инженер ступени 9, как правило, только закончил образование и начинает осваивать принципы инженерии ПО. Работает под постоянным контролем.	Нет
10	Инженер 10-й ступени имеет базовое представление об инженерии ПО. Как правило, недавно получил образование или имеет 1–2 года опыта работы. Может выполнять работы при ограниченном контроле.	Вводный во всех областях. Продвинутый в 3 областях.
11	Твердые знания в инженерии ПО, может работать самостоятельно. Работал в нескольких завершенных проектах и имеет опыт участия в каждом этапе цикла разработки ПО, необходимого для выпуска продукта.	Вводный во всех областях. Продвинутый в 6 областях. Ведущий в одной области.

Таблица 16.3 (продолжение)

Ступень лестницы	Описание	Охваченные области знаний (СКА)
12	Инженер 12-й ступени успешно участвует во всех аспектах небольших и крупных проектов, делая весомый вклад в их эффективность. Зарекомендовал себя грамотными техническими решениями и компетентностью в вопросах проекта. Вносит инновации, обеспечивает устойчивый уровень работ, превышая объем поставленных задач. Обычно осуществляет техническое руководство других сотрудников или надзор за ними.	Вводный во всех областях. Продвинутый в 8 областях. Ведущий в 3 областях.
13	Инженер на ступени 13 – настоящий руководитель, способный анализировать внешние и внутренние стороны проекта и обеспечивать грамотные и правильные решения. Полностью владеет всеми аспектами своего проекта и делает уникальный вклад.в работу. Решения этого инженера приносят существенную прибыль компании и обеспечивают общее устойчивое положение.	Вводный во всех областях. Продвинутый в 8 областях. Ведущий в 5 областях. Мастерский в 1 области.
14	Инженер ступени 14 – это крупный инженерный ресурс для компании. Справляется с серьезными проблемами, принимает решения по ключевым задачам и структуре компании. Известен многим инженерам ПО внутри и вне компании по конкретным достижениям, которые реально способствовали развитию инженерии ПО. Объем знаний охватывает всю отрасль. Достижение этой ступени требует посвятить свою деятельность инженерии ПО, в том числе за пределами Construx.	Намеренно не определены.
15	Незаменим для успеха компании. Постоянно разрабатывает и выдает прорывные продукты мирового класса. Практикующими инженерами внутри и вне компании признан ведущим экспертом в инженерии ПО. На него возлагается основная ответственность за раз-	Намеренно не определены.

Ступень лестницы	Описание	Охваченные области знаний (СКА)
	работку методик компании. Вносит постоянный и разнообразный вклад в работу всей отрасли. Работа на этой ступени требует посвятить всю деятельность инженерии ПО (и не только в Construx) и обуславливает признание достижений, которые выходят за пределы освоенной области.	

Развитие карьеры на основе продвижения по лестнице

Посмотрим на примере продвижения технически «подкованного» инженера со ступени 10 на ступень 12, каким образом лестница профессионального развития в Construx обеспечивает карьерный рост. Пусть целевыми областями знаний для него будут инструменты и методики инженерии, проектирование и создание ПО.

На рис. 16.2 показаны требования к инженеру для достижения ступени 10. Черные квадратики показывают достигнутый уровень способностей в каждой области знаний.

Как видно из рис. 16.3, для достижения ступени 11 инженер растет «вглубь», выходя на уровень ведущего в области создания ПО, и «в ширину», достигая продвинутого уровня в нескольких других областях.

На рис. 16.4 показано, что для выхода на ступень 12 инженер достигает ведущего уровня в области проектирования и инструментария и методик инженерии и продвинутого уровня дополнительно в двух СКА.

	Области знаний									
	Управление конфигурацией	Создание	Проектирование	Управление инженерией	Инженерный процесс разработки	Обслуживание	Качество	Требования к ПО	Тестирование	Инструментарий и методики инженерии
Вводный	■	■	■	■	■	■	■	■	■	■
Продвинутый		■							■	■
Ведущий										

Рис. 16.2. *Пример требований лестницы профессионального развития для выхода на ступень 10*

	Области знаний									
	Управление конфигурацией	Создание	Проектирование	Управление инженерией	Инженерный процесс разработки	Обслуживание	Качество	Требования к ПО	Тестирование	Инструментарий и методики инженерии
Вводный	■	■	■	■	■	■	■	■	■	■
Продвинутый		■	■	■			■		■	■
Ведущий		■								

Рис. 16.3. *Пример требований лестницы профессионального развития для выхода на ступень 11*

	Области знаний									
	Управление конфигурацией	Создание	Проектирование	Управление инженерией	Инженерный процесс разработки	Обслуживание	Качество	Требования к ПО	Тестирование	Инструментарий и методики инженерии
Вводный	■	■	■	■	■	■	■	■	■	■
Продвинутый		■	■	■	■		■	■	■	■
Ведущий		■	■							■

Рис. 16.4. *Пример требований лестницы профессионального развития для выхода на ступень 12*

Лестница профессионального развития предоставляет несколько путей карьерного роста, давая возможность сотрудникам выбрать области знаний, на которых они намерены сконцентрироваться. Этим обеспечивается гибкость и структура, поскольку сотрудник выбирает свой карьерный путь, руководствуясь требованиями уровня областей знаний. Из рис. 16.2–16.4 можно увидеть, что разработчик сосредоточился на знаниях инструментария и методики инженерии, проектирования и создания ПО. В другом случае можно говорить о совершенствовании в управлении инженерией, процессе разработки и требованиях ПО. Нацеленный на качество инженер мог бы выбрать для совершенствования аналогичную область, процесс разработки и тестирование. Лестница профессионального развития Construx является универсальным средством определения карьерного продвижения независимо от узкой специализации и ориентации инженера.

Требования СКА для различных уровней способностей

Лестница развития содержит четыре уровня способностей, но мастерский уровень по определению не может быть сформулирован, поэтому обычно лестница описывается матрицей 10×3: десять столбцов – области знаний и три строки – уровни способностей. В каждой области матрицы определены элементы, которые должны соответствовать ее требованиям, такие как чтение, семинары и опыт работы. Всего лестница профессионального развития на данный момент содержит около тысячи конкретных требований.

Области управления инженерией в Construx является хорошим примером широты и глубины требований ко всем уровням способностей – вводному, продвинутому и ведущему. В табл. 16.4 показано, что следует прочитать и какой практический опыт накопить, чтобы выйти на вводный уровень знаний по управлению инженерией ПО.

Как видно из табл. 16.5, для достижения продвинутого уровня в этой области необходимы намного более интенсивные усилия.

Таблица 16.4. Требования к вводному уровню знаний по управлению инженерией ПО

Действия	Требования
Чтение	Необходимо *проанализировать*[a] работы: • Чарльза Фишмана [44] • Стива Макконнелла [84] • «Software Engineering Code of Ethics and Professionalism» (Кодекс профессиональной этики инженерии ПО), ACM/IEEE-CS[b] Следующие материалы необходимо *изучить информативно*: • Работу Алана Дейвиса [34] • Книгу Иана Соммервилля [127], часть 1 и главы 22–23 • Книгу Роджера Прессмана [111], часть 4

[a] В рамках лестницы профессионального развития различаются аналитическое и информативное чтение. Подробно это различие разъясняется в работе Стива Макконнелла [85].

[b] См. *www.construx.com/profession*.

Таблица 16.4 (продолжение)

Действия	Требования
Опыт работы	• Разобрать план проекта • Знать методики оценки • Планировать и отслеживать работу сотрудников
Семинары	Нет
Сертификация	Нет
Участие в отраслевой деятельности	Нет

Таблица 16.5. Требования к продвинутому уровню знаний по управлению инженерией ПО

Действия	Требования
Чтение	Необходимо *проанализировать* следующие работы: • Книгу Фредерика Брукса [20] • Работу Демарко и Листера [36] • Работу [94] • Книгу Стива Макконелла [83] Следующие материалы необходимо *изучить информативно*: • Работу Уейта Гиббса [48] • Работу [94]
Опыт работы	• Участвовать в качестве эксперта артефактов управления проектом • Участвовать в создании устава проекта • Участвовать в разработке плана проекта • Участвовать в разработке оценки проекта; владеть методами индивидуальной оценки снизу вверх; возглавить оценочную работу • Владеть методами индивидуальной отчетности по состоянию дел проекта; формировать еженедельный отчет состояния проекта • Владеть планированием работ (включая способы разделения работы на участки, оценку и управление наработанной стоимостью)
Семинары	• Справочник по управлению выживанием проектов (2 дня)

Действия	Требования
	• Быстрая разработка ПО (2 дня) • Оценка ПО (2 дня)
Сертификация	Нет
Участие в отраслевой деятельности	Нет

Переход с продвинутого на ведущий уровень области знаний не столь жестко прописан, как переход с вводного на продвинутый уровень. Подробные требования вырабатываются совместно сотрудником, его куратором и руководителем (менеджером). На ведущем уровне помимо чтения, семинаров и опыта работы от сотрудника может потребоваться получение признанного отраслевого квалификационного сертификата и участие в отраслевой работе в качестве преподавателя на семинарах, докладчика на конференциях и т. д.

В табл. 16.6 подробно описаны действия и масштаб работы, необходимые для такого перехода.

Таблица 16.6. Требования к продвинутому уровню знаний по управлению инженерией ПО

Действия	Требования
Чтение	Материалы для чтения подбираются индивидуально. Целевая область и выбор конкретных материалов определяются совместно с куратором и указываются в плане профессионального развития. Для перехода с продвинутого на ведущий уровень в каждой СКА ориентировочно требуется прочитать около тысячи страниц.
Опыт работы	• Возглавлять планирование, оценку и отслеживание работы по значительному проекту; выработать устав проекта с ТЭО; выработать план проекта для крупного и небольшого проекта; создать рабочий план значимого проекта; определять цикл существования проекта, набор характеристик и планирование; владеть формальными методами управления рисками проекта; владеть формальными методами управления проблемами проекта; владеть методами сбора ретроспективных данных. • Владеть групповыми методами оценок (например, расширенным методом дельфийского оракула), методами оценки аналогии и параметрической оценки; создать оценку проекта сверху вниз на стадии формулировки проекта.

Таблица 16.6 (продолжение)

Действия	Требования
	• Создать план-график работ (или этапов) значимого проекта; создать подробный план-график этапа; владеть методами критического пути и критической цепочки для создания расписаний (Ганнт и PERT); владеть методами отчетности о состоянии проекта. Участвовать в консультационной работе/инструктировании по управлению инженерией ПО.
Семинары	• Эффективное управление проектами ПО (3 дня) • Управление рисками (2 дня) • Подряд сторонних организаций для проекта (2 дня)
Сертификация	• Получить свидетельство сертифицированного профессионального разработчика ПО компьютерного общества IEEE[a] • Получить сертификат профессионала управления проектами PMI[b]
Участие в отраслевой деятельности	На ведущем уровне ожидается существенный вклад сотрудников в работу Construx и потенциально в отрасль в целом. Примеры участия в работе отрасли: • Разработать вечерний, воскресный или институтский курс в этой области; вести вечерние, воскресные или институтские семинары • Преподавать на семинарах в Construx или университете • Участвовать в отраслевом комитете, дискуссионной группе, совете по стандартам и т. п. • Сделать доклад на конференции • Опубликовать статью в ведущем издании или обозрении • Опубликовать статью в издании второго эшелона • Отрецензировать рукопись книги • Рецензировать статьи для IEEE Software или аналогичного издания • Активно участвовать в инструктаже и кураторстве других сотрудников Construx по избранной ведущей области

[a] См. *www.computer.org/certification*.

[b] См. *www.pmi.org*.

Выводы, сделанные по результатам лестницы профессионального развития

Версия 1.0 лестницы профессионального развития была внедрена в Construx в 1998 г. и сразу после этого опубликована. Ее эволюция нашла отражение в версии 2.0, вышедшей в начале 2002 г. для внутреннего использования. За это время нами сделан ряд важных выводов о внедрении и поддержке функционирования лестницы профессионального развития.

Структурные и культурные усиления

Чтобы обеспечить успешное профессиональное развитие в организации, программа лестницы должна быть встроена в ее организационную культуру. Мы определили разнообразные меры, чтобы гарантировать принятие программы сотрудниками и достижение желаемых результатов.

Включение конкретных мер усиления, внедренных в Construx, не обязательно для успешной реализации лестницы профессионального развития, но важно определить структурные и культурные стимулирующие элементы, которые сработают в данной организации.

Конкретные меры стимулирования, внедренные в Construx, включают:

План профессионального роста (Professional Development Plan, PDP). Это механизм планирования, отслеживания и документирования продвижения сотрудника по лестнице профессионального развития. Каждый PDP включает кратко- и долгосрочные цели сотрудника и конкретные действия и факторы (чтение, семинарские занятия, опыт работы и другую профессиональную деятельность), которые выполняются (формируются) в промежутках между отчетными датами.

В цели PDP практически всегда входит повышение категории персонала на период от одного до пяти лет. План определяет работу, которую должен выполнить сотрудник, чтобы добиться повышения. На период более года эта работа не формулируется в окончательном виде, но составляется общий план. Когда до повышения остается меньше года, работа расписывается подробно на каждый месяц для куратора, сотрудника и его менеджера. План дает объективные и согласованные критерии повышения по службе для всей компании.

Программа кураторов. Задача куратора – руководство и поддержка продвижения инженера по лестнице. Весь инженерный состав Con-

strux разрабатывает и обсуждает свои планы с кураторами. Институт кураторов позволяет сформировать лестницу профессионального развития для каждого отдельного сотрудника с ориентацией на него планов профессионального роста. Чтобы выполнить требования, предъявляемые к знаниям сотрудника, куратор и сотрудник встречаются 6–8 раз в год и чаще, когда до повышения сотрудника остается менее полугода.

Активное взаимодействие менеджера и куратора абсолютно необходимо для выполнения всех действий, предусмотренных планом роста. С целью поддержки этого процесса руководитель подразделения и куратор должны подписать план профессионального роста сотрудника. На встречах с куратором отслеживается выполнение этого плана и, если необходимо, корректируется срок повышения.

Одной из целей института кураторов является выработка у инженеров вкуса к самостоятельности в дальнейшем профессиональном развитии. Инженеры на 12-й ступени и выше должны проявлять самостоятельность, и кураторы им не нужны, если только они специально не просят об их назначении или собираются добиваться нового повышения категории.

Свидетельства профессионального развития. Construx отмечает разнообразные этапы и события в процессе профессионального развития сотрудников: повышение категории, получение профессиональных сертификатов, первый проект в роли ведущего, первые проведенные занятия, первые опубликованные работы и другие значительные события. Каждый разработчик получает памятную пластинку с выгравированной надписью, обозначающей этапные достижения. Этим отмечаются существенные события и подчеркивается важное место, которое Construx отводит профессиональному развитию.

Программа обучения. Помимо обычного, текущего обучения, которое незримо идет при разработке ПО, Construx выделяет 10–12 дней в год для целенаправленных занятий. Для специалистов с небольшим опытом они обычно проводятся в форме семинаров и конференций. На ступени 12 и выше обучение может заключаться в подготовке выступлений на конференциях, участии в комитетах по стандартам, организации специальных групп по отдельным вопросам и другой профессиональной деятельности.

Структура заработной платы. Было установлено, что традиционная система вознаграждений за труд должна быть перестроена, чтобы содействовать целям профессионального развития; в противном случае задачи проекта превалировали бы над задачами развития. Повышения в должности, зарплата и трудовые показатели, основанные на механизме лестницы развития, дают возможность прочно закрепить за ним особое место в структуре организации.

Каждой ступени лестницы профессионального развития соответствует в точности один уровень оплаты труда, при этом в Construx ступень каждого сотрудника и зарплата на каждой ступени являются открытой информацией. Поэтому у сотрудников есть очевидный стимул перейти на более высокую ступень, поскольку такой переход означает новый уровень зарплаты.

Группы по инженерии ПО. Эти группы (Software Engineering Discussion Groups, SEDG) представляют собой место для обсуждения и обмена опытом по инженерии ПО. В них под руководством ведущего группы обсуждаются и анализируются книги, входящие в перечень для чтения в рамках лестницы профессионального развития. Такие группы есть на ступенях 9, 10 и 11. Инженерам на ступени 12 рекомендуется участвовать в работе групп с целью поделиться опытом и знаниями с коллегами, находящимися на более низких ступенях.

Признание ступени 12. Достижение ступени 12 – полного профессионального статуса в Construx – является значительной вехой карьеры инженера ПО в компании. В ознаменование этого достижения Construx премирует добившихся этого сотрудников в сумме разницы годовой зарплаты на ступенях 11 и 12, устраивает прием в их честь, публикует информацию в местной газете деловых кругов и вывешивает портрет в фойе здания компании. Этим подчеркивается важность, которую компания придает профессиональному развитию.

Как встроить опытных инженеров в лестницу профессионального развития

Оказалось, что и новых сотрудников, практиковавших в других компаниях, также нужно как-то поместить на лестницу развития. Многие из претендентов на должности имеют значительный опыт работы в отрасли, но не отвечают другим требованиям нашей лестницы развития. Если при-

нимать их на зарплату ступени 10 или 11, то мы не сможем конкурировать с другими работодателями. В практическом плане нам бы не удалось привлечь старший персонал. Поэтому важно было разработать механизм перевода нового сотрудника на ступень 12, не разрушая цельности лестницы развития и не преуменьшая достижения других сотрудников Construx, которые прошли путь к ступени 12 в фирме.

Чтобы решить эту задачу, была создана «переходная ступень 12». Поступающий на работу опытный инженер сразу находится на ступени 12, но в течение года или менее обязан восполнить недостающие элементы этой ступени лестницы – в основном интенсивным чтением. В течение данного года сотрудник ежемесячно встречается со своим куратором для обсуждения выполненных им работ и всех вопросов продвижения по лестнице. Когда недостающие элементы восполнены, сотрудник получает официальное признание как инженер ступени 12.

Преимущества лестницы профессионального развития

Пять лет работы в рамках версии 1 и 2 лестницы профессионального развития позволили реализовать множество преимуществ.

- *Ускоренное профессиональное развитие.* Мы достигли нашей главной цели – повышения профессионализма нашего инженерного персонала, и его развитие идет чрезвычайно быстро. Во время собеседований с претендентами на занятие вакантных должностей мы обычно видим, что внутренние стандарты в Construx для инженера с двух-трехлетним стажем сравнимы с требованиями, предъявляемыми к самому опытному персоналу во многих других компаниях.
- *Командный потенциал.* Стандартизировав наши области знаний, мы смогли получить общую основу практического опыта и знаний. Это улучшило взаимопонимание между сотрудниками и обеспечило высокую эффективность работы в проектах в особой роли лидеров.
- *Хорошо воспринимаемые критерии повышения в должности.* Было установлено, что технический персонал компании высоко оценивает открытость механизма повышения. Сотрудники видят, что могут контролировать свой карьерный рост и что перед ними действительно открывается перспектива, а не просто возможность получить рабочее место.

- *Привлечение персонала.* Опыт работы с областями знаний внутри компании позволяет непосредственно оценивать кандидатов на инженерные должности. Мы можем оценить способности кандидатов в каждой области знаний. Оценки кандидатов в проектах различными нашими сотрудниками оказываются весьма согласованными.
- *Отказ в приеме на работу.* Разработанный нами механизм лестницы развития требует определенной подотчетности, что не очень нравится некоторым кандидатам для приема на работу. Когда в процессе поиска кандидатам прямо указывается на требования профессионального развития, это позволяет отсеять многих из них, чьи намерения в связи с инженерией ПО нельзя считать серьезными.
- *Управление квалификационными навыками.* Матрица 10×3 областей знаний является естественным структурированным методом отслеживания возможностей и способностей персонала нашей компании.
- *Моральное состояние и удержание сотрудников.* Хотя лестница профессионального развития является лишь частью всей картины, мы убеждены, что она вносит огромный вклад в повышение морального духа и стабильности коллектива. На момент написания этой книги фирма Construx третий год подряд стала финалистом конкурса, проводимого изданием *Washington CEO Magazine*, на звание «Малого предприятия, где лучше всего работать». В течение более трех лет ни один сотрудник не ушел из компании по собственному желанию.

Использование лестницы профессионального развития в других компаниях

Лестница профессионального развития используется в Construx как гибкий и структурированный механизм поддержки развития карьеры. Определяя и развивая способности в различных областях знаний, Construx обеспечивает широту и глубину квалификации персонала в соответствии с инженерным подходом к разработке ПО. Гибкость механизма дает возможность нашему персоналу выбирать продвижение по лестнице, отвечающее их собственным интересам.

Construx предъявляет более жесткие требования к способностям, чем большинство компаний в соответствии с нашей корпоративной миссией – «развивать науку и искусство инженерии ПО». Большинство наших инженеров, находящихся на высоких должностях, занимаются консультированием, преподаванием, возглавляют проекты, а это значит, что большее

число наших сотрудников способны выйти на ведущий уровень по сравнению с другими компаниями.

Мы пришли к выводу, что опора на области знаний, сформулированная в SWEBOK, дала нам существенные выгоды. Выстраивание областей знаний в виде матрицы 10×3 означает, что лестницу профессионального развития можно легко перестроить, добавляя или убирая уровни способностей, области знаний и специальные требования для удовлетворения нужд конкретной организации. Используя структуру матрицы 10×3, можно формировать на основе лестницы особые пути карьерного роста, например, менеджер проектов, бизнес-аналитик, разработчик, тестировщик ПО, обеспечивая структуру и направление развития внутри организации, которая не склонна разъяснять персоналу внутренний механизм функционирования лестницы.

Возможны некоторые отличия при внедрении лестницы в других компаниях, но ее структура и основные понятия могут использоваться в любой организации. В каждом отдельном случае фундаментальная основа лестницы обеспечивает высокую степень единства и целостности подхода.

ЧАСТЬ ЧЕТВЕРТАЯ

Индустриальный профессионализм

ГЛАВА СЕМНАДЦАТАЯ

Построение профессии

Инженерная работа может дать настоящее удовлетворение в разных сферах, но особенно – принося практическую пользу человечеству и служа его благосостоянию.

Ванневар Буш (Vannevar Bush)

В отношении инженеров сложился свой стереотип, очень похожий на стереотип программиста. Их считают скучными и нудными, однако именно эти нудные и скучные люди являются авторами некоторых самых впечатляющих творений нашего времени. Прогулка человека по Луне, взгляд на внутреннее строение космоса с помощью телескопа Хаббл, полеты на современных самолетах, поездки на автомобилях, работающих почти безупречно, подключение к интернет-сайтам по всему миру, удовольствие от просмотра фильмов в домашнем кинотеатре – все эти чудеса технологии стали в основном результатом инженерных достижений, практического воплощения научных принципов.

Необходимость инженерии

Исторически профессиональная инженерная наука сформировалась в ответ на угрозы общественной безопасности. Хотя безопасность современных мостов воспринимается как данность, в 60-х годах XIX века в Америке ежегодно рушилось более 25 мостов [45]. Рухнувшие мосты и их жертвы привели к формированию строгого инженерного подхода к проектированию мостов и их строительству. В Канаде инженерный фольклор утверждает, что мост, рухнувший в Квебеке в 1907 г., стал катализатором установления более высоких стандартов во всех областях инженерной

деятельности Канады, что символизирует сегодня церемония железного кольца.[1] (Эта церемония подробно описана в главе 19.) Инженеры в Техасе стали получать лицензии только после взрыва в бойлерной начальной школы в 1937 г., унесшего жизни более 300 детей.[2] Агрегат, ставший причиной взрыва, сегодня заменен программным обеспечением.

Инженерная профессия отличается от других тем, что, например, врачи, зубные техники, бухгалтеры и адвокаты в основном предоставляют свои услуги конкретным физическим лицам или в некоторых случаях юридическим лицам. Инженеры же создают *предметы*, а не предоставляют услуги частным лицам. Они несут ответственность перед обществом в целом, а не перед отдельными гражданами. В этом смысле разработчики ПО сегодня больше похожи на инженеров, чем на представителей других профессий. В мире ПО пока не случилось своего обрушения моста, как в Квебеке, или взрыва бойлерной, как в техасской школе. Но потенциальная возможность сохраняется; это известно любому посетителю форума, посвященного риску при использовании компьютеров и связанных с ними систем.[3] ПО уже стало виной многомиллионных убытков в ряде случаев, начиная со смешных до закончившихся смертельным исходом. Цутому Шимомура оставил свой автомобиль на парковке аэропорта в Сан-Диего 29 февраля 1992 г. Когда он вернулся через 6 дней, счет за стоянку составил более 3 тысяч долларов. ПО парковки не распознало дату 29 февраля как правильную.[4] В январе 1990 г. из-за ошибки ПО за 9 часов было блокировано примерно пять миллионов телефонных звонков. Первый полет корабля многоразового использования был отложен на два дня из-за трудноуловимой ошибки программиста. Космический зонд «Маринер-1» был потерян на пути к Венере из-за ошибки в программировании управления навигацией. В Лондоне автоматизированная система отправки бригад скорой

---

[1] Я называю эти истории фольклором, т. к. несколько канадских инженеров рассказывали мне, что стальное кольцо традиционно считается сделанным из остатков того самого моста, рухнувшего в Квебеке в 1907 г. Опубликованные сведения о церемонии стального кольца в Канаде не содержат упоминаний о квебекском мосте. В самой церемонии мост, возможно, и упоминается, но это тайная церемония.

[2] *The New York Times*, 3 мая 1999 г.

[3] Подписку на доступ к этому форуму можно оформить по электронной почте: *risks-request@csl.sri.com* или через Usenet на *comp.risks*.

[4] Все примеры в данном пункте взяты из книги Ньюмана Питера Дж. (Neumann Peter G.) «Computer-Related Risks» (Риски, связанные с компьютерами), Reading, MA: Addison-Wesley, 1995.

помощи по вызовам была введена в действие раньше, чем была готова, и полностью рухнула, что вызвало задержки до 11 часов. Новая система отправки бригад стала причиной 20 смертных случаев. Самолет, выполнявший рейс № 655 иранских авиалиний, был сбит системой «Эгида» (щит Зевса) американского авианосца «Винсенн» в 1988 г. Погибло 290 человек. Поначалу ошибку записали на счет оператора, но позднее некоторые специалисты посчитали причиной происшествия плохой дизайн пользовательского интерфейса системы «Эгида».

Искусство и инженерия

Применение математики и научных принципов в инженерной практике делает ее открытой для обвинений в сухости, в том, что они выхолащивают инженерное дело. Та же самая критика звучит и в отношении инженерии ПО. Насколько справедливы эти утверждения? Инженерия связана со всеми аспектами конструирования и проектирования, в том числе и с эстетическими. Инженерный дизайн не ограничивается формой и цветом. Инженеры проектируют все – от электронных плат и несущих перекрытий до аппаратов для посадки на Луне. Как замечает Сэмюэль Флорман (Samuel C. Florman) в книге «Existential Pleasures of Engineering» (Экзистенциальные удовольствия инженерии), «творческое проектирование – вот главная миссия профессионального инженера». Сравним два хорошо известных архитектурных творения: кафедральный собор в Реймсе, законченный примерно в 1290 г. (рис. 17.1), и здание Сиднейской оперы (рис. 17.2), построенное в 1973 г. При проектировании собора рассчитывали на материалы, свойства которых были более или менее известны в то время.

Здание оперы было построено на 700 лет позже собора. Как видно из рис. 17.2, стилистически оно кардинально отличается от собора в Реймсе. Архитекторы взяли современные материалы (сталь, железобетон) и применили инженерные методы, в том числе и компьютерное моделирование, чтобы определить минимальное количество материалов, которое можно использовать, не подвергая здание риску разрушения.

Предпочесть то или иное здание – дело вкуса. Но какое здание можно было построить – это инженерный вопрос. Современные архитекторы могли бы выстроить еще один Реймский собор, но строители XIII века не смогли бы построить Сиднейскую оперу. Причина невозможности постройки этого здания в XIII веке – не отсутствие искусства архитектуры, а нехватка инженерных возможностей. Каждый может припомнить ужас-

Рис. 17.1. *Реймский кафедральный собор, г. Реймс, Франция. Пример искусства во времена неразвитой инженерии*

Рис. 17.2. *Сиднейская опера, г. Сидней, Австралия. Пример зависимости искусства от инженерных решений*[1]

[1] Рисунки 17.1 и 17.2 взяты из сборника фотографий IMSI's MasterClips©/MasterPhotos©, 1895 Francisco Blvd. East, San Rafael, CA 94901-5506, USA.

ные сооружения, где искусство проиграло инженерной экономии или, наоборот, где не принималась во внимание эстетика. Инженерия, лишенная искусства, может быть ужасной, но искусство без инженерии может оказаться невозможным. Не инженерия, а ее нехватка ограничивает художественные изыски.

То же самое справедливо и для современных систем ПО. Уровень инженерного мастерства определяет объем систем ПО, которые могут быть созданы, насколько легко ими можно будет пользоваться, их быстродействие, количество возможных ошибок в них, а также то, насколько свободно они будут взаимодействовать с другими системами. В ПО содержится множество эстетических элементов, а разработчики ПО отсутствием стремления к артистизму не страдают. Чего иногда не хватает при разработке ПО, так это инженерных методик, которые позволяют реализовать наши самые грандиозные эстетические устремления.

Инженерные дисциплины достигают зрелости

Катастрофы в давно сложившихся областях инженерии ускорили профессионализацию инженерных работ. Разумеется, полноценная инженерная дисциплина не может быть внедрена в практику простым волевым усилием. Мэри Шоу (Mary Shaw) из Университета Карнеги Меллона описала поэтапный процесс, который проходит инженерная область, пока не достигнет профессионального уровня (рис. 17.3).

На стадии *ремесла* достойная работа выполняется талантливыми самоучками-любителями. На вооружении ремесленников интуиция и простая сила, с помощью которых они создают свои творения, будь то мосты, электроприборы или компьютерные программы. Некоторые изделия предназначены для продажи, но большинство создается исключительно для личного пользования. У ремесленников есть лишь слабое представление о мас-

Рис. 17.3. *Развитие дисциплины от ремесла к профессиональной инженерной практике. Источник: «Prospects for an Engineering Discipline of Software» (Перспективы превращения ПО в инженерную дисциплину» [124]*

совом производстве для продажи, а иногда оно и вовсе отсутствует. В большинстве случаев они находят имеющимся в их распоряжении материалам нестандартное применение. Практическая сфера развивается хаотично, нет систематического образования, и никто не обучает ремесленников применению самых эффективных методов.

Гражданское строительство (возведение мостов, акведуков) в Древнем Риме было на стадии ремесла, как и применение компьютеров в вычислениях в начале 50-х и 60-х годов XX века. Многие проекты ПО сегодня все еще самым причудливым образом расходуют наличный ресурс (рабочее время персонала) и функционируют на уровне ремесла. В какой-то момент спрос на изделия начинает превышать возможности самого ремесленника-одиночки и диктует расширение производства, что начинает влиять на формирование дисциплины. По мере усложнения рецептов изготовления начинают складываться письменные своды эвристических и процедурных правил.

На *коммерческой* стадии работники тщательнее подходят к ресурсам, необходимым для обеспечения производства. Эта стадия отмечена более выраженной экономической направленностью, и стоимость изделий может стать источником затруднений.

Практики обучаются с целью обеспечить постоянство качества производимых ими изделий. Они варьируют параметры и проверяют, что работает, а что нет, и благодаря этому производственные методы систематически совершенствуются.

Реймский собор был построен в то время, когда гражданское строительство находилось на коммерческой стадии развития. В области ПО многие организации, находящиеся на коммерческой стадии, вышли на достойный уровень качества продукции и производительности, привлекая тщательно отобранный и обученный персонал. Эти организации опираются на освоенные методики и модифицируют их крайне незначительно, преследуя цель выдавать продукт лучшего качества и поднять выполнение проектов на более высокий уровень.

Некоторые проблемы на коммерческой стадии невозможно решить методом проб и ошибок. Поэтому, если экономические ставки достаточно высоки, начинает складываться соответствующая наука. По мере формирования науки в ней разрабатываются теории, которые обеспечивают коммерческую практику, и именно на этом этапе сфера деятельности достигает стадии *профессиональной инженерии*. В этот момент прогресс наступает как в результате применения научных принципов, так и вследст-

вие практических экспериментов. Занятые в данной сфере люди на этой стадии должны быть хорошо подготовлены в своей профессии как теоретически, так и практически.

Наука для разработки ПО

Наука ПО годами отставала от развития коммерческого ПО. В 1950–1960 гг. разрабатывались грандиозные системы ПО, такие как противоракетная система Sage, система бронирования билетов на авиалиниях Sabre, операционная система IBM OC/360. Коммерческая разработка этих систем продвигалась значительно быстрее, чем соответствующая научная основа, но в инженерных дисциплинах практические приложения обычно развиваются быстрее науки. Аэродинамический профиль крыла, который позволяет летать самолетам, был создан сразу же после того, как было «доказано», что никакой аппарат тяжелее воздуха летать не может.[1] Развитие термодинамики последовало за изобретением паровой машины. Во времена Джона Реблинга (John Roebling) прочность стальных тросов еще не была изучена, поэтому различные секции моста были спроектированы с запасом прочности от 6 до 1. Этот запас был порожден инженерным чутьем, заменявшим теоретическое знание.

Наука, обеспечивающая разработку ПО, еще не столь хорошо сформулирована, как физика, на которую опирается гражданское строительство. На самом деле она даже не считается «естественной». Герберт Саймон (Herbert Simon) называл ее «наукой об искусственном» [125] – это области знаний компьютерной науки, математики, психологии, социологии и науки управления. Несколько организаций, связанных с производством ПО, регулярно применяют теории этих наук в своих проектах, но до их всеобщего применения к проектам ПО еще далеко.

Но требуем ли мы от науки ПО действительно нужных результатов? Во многих программах – системах управления складскими запасами, расчета зарплаты и учета персонала, формирования главной бухгалтерской книги, проектирования операционных систем, управления базами данных, трансляторах языков программирования (перечень почти бесконечен) – одни и те же базовые приложения были написаны столько раз, что такие системы не должны требовать сильного напряжения уникальной

[1] Кристофер Александер (Christopher Alexander), цитируется по книге Роберта Л. Гласса [52].

изобретательской мысли, как это может показаться. Мэри Шоу указывает, что в сложившихся инженерных областях обычное проектирование включает решение стандартных задач и повторное применение готовых решений. Очень часто такие «решения» сформулированы в виде равенств, аналитических моделей или готовых блоков-компонент. Время от времени возникают неординарные задачи проектирования, но главная работа инженеров состоит в применении стандартных методик к знакомым задачам.

Мир ПО еще только на пути формирования своих «стандартных решений», предназначенных для среднестатистического пользователя. Многие наработки проектов ПО потенциально могут быть применены многократно, огромное их количество имеет больший потенциал повышения качества и производительности, чем его наиболее часто применяемый искусственный объект (артефакт) – исходный код. Вот краткий перечень таких искусственных объектов, ориентированных на многократное использование [64]:

- Архитектура сама по себе и процедуры проектирования ПО
- Схемы проектирования
- Технические требования сами по себе и процедуры их разработки
- Элементы интерфейса пользователя и процедуры его проектирования
- Сметные стоимости и процедуры их формирования
- Данные планирования, планы проектов и процедуры планирования
- Планы испытаний, схемы тестов, данные тестов и процедуры тестирования
- Процедуры технических экспертиз
- Исходный код, процедуры компоновки и интеграция
- Процедуры управления конфигурацией ПО
- Отчеты после завершения и экспертиз проектов
- Организационные структуры, состав групп разработчиков и процедуры управления

На сегодня совсем мало проектных наработок собрано в форме, легко применимой любой организацией. Наука еще не обеспечила разработку ПО набором формул, описывающих успешное управление проектом, или методикой создания удачного продукта ПО. Возможно, этих описаний никогда и не будет. Но наука вовсе не должна состоять из формул и математических выражений. В своей работе [76] Томас Кун (Thomas Kuhn) указывает, что научная парадигма может состоять из набора решенных задач. Много-

кратно используемые артефакты проектов образуют набор решенных проблем – требований, проектирования, планирования, управления и т. д.

Зов инженерии

Артур Чарльз Кларк говорил, что «любая достаточно развитая технология неотличима от магии». Технология ПО достаточно развита, и общество уже заворожено ею. Люди не осознают, с какими рисками связаны программные продукты, или финансовые риски, с которыми связаны проекты разработки ПО. Как жрецы этой могущественной магии разработчики ПО несут профессиональную ответственность за ее мудрое использование.

Инженерный подход к проектированию и реализации зарекомендовал себя за долгие годы как устраняющий (почти) риски общественной безопасности, одновременно воплощая самые дерзкие устремления человеческого духа. Какими бы ни были цели – безопасность, эстетика или экономика, – подход к ПО как к инженерной дисциплине – это эффективный способ поднять его разработку до уровня настоящей профессии.

ГЛАВА ВОСЕМНАДЦАТАЯ

Школа жизни

Природные способности как растения: им нужен уход в виде обучения, но само учение дает слишком много простора (зато часто рождает самомнение (амбиции), если не ограничено опытом).

Френсис Бэкон

Выше уже говорилось, что большинство разработчиков ПО осваивали профессию в школе жизни. Опыт, может быть, и хороший учитель, но уж очень медленный и дорогостоящий.

Опытные разработчики сетуют, что в колледжах не прививают навыков, необходимых для эффективной работы по профессии. Это подтверждается анализом демографической структуры и системы обучения разработчиков ПО. В главе 4 утверждается, что в североамериканских университетах преподается скорее компьютерная наука, а не инженерия ПО. Многие следствия этого вывода пока оставались в стороне, но теперь настало время поговорить и о них.

Отрасль все меньше и меньше удовлетворяют выпуски специалистов из университетов. Кейперс Джоунз указывал, что с середины 80-х годов прошлого века в крупных корпорациях в США было больше своих преподавателей по разделам инженерии ПО, чем во всех университетах вместе взятых [65]. Многие из этих корпораций предлагали более полные программы курсов по инженерии ПО, чем практически любой университет.

В компании «Боинг» изучили программы курсов инженерии ПО более чем 200 университетов США [46] с целью выявить те из них, которые могут обеспечить навыки и образование, необходимые для успешной работы в «Боинге». Оказалось, что лишь около половины программ были аттестованы Советом по аккредитации в области компьютерных наук (CSAB),[1]

и лишь половина из них готовила выпускников, отвечающих требованиям «Боинга». Некоторые программы были больше ориентированы на практическую работу, но студентов обучали инженерии ПО под видом компьютерной науки, что размывало реальное представление о ней.

В то же время на инженерные специальности «Боинг» принимал заявления о приеме на работу от выпускников, прошедших обучение по любой программе, аттестованной Советом по аттестации инженерии и технологии (ABET), и не принимал заявления от выпускников по аттестованным программам. Это приводит к выводу, что программы обучения по компьютерным наукам и стандарты аттестации имеют для отрасли сомнительную ценность. Значительная доля программ не отвечает запросам отрасли. В противоположность этому ценность программ инженерного обучения и аттестации настолько упрочилась, что такие компании, как «Боинг», могут смело принимать на работу выпускников инженерных отделений, не проводя самостоятельный отбор университетов.

Сомнительная согласованность образования в области компьютерной науки и нужд отрасли дает одно возможное объяснение упадку и застою, который наблюдается в количестве выпускников по специальности «компьютерная наука» с 1985 по 1997 гг. Как видно из рис. 18.1, оно упало с высшей отметки примерно в 42 тысячи выпускников ежегодно в 1985 г. до низшего уровня примерно в 24 тысячи с 1990 по 1997 гг.

В последние годы (1998–2000 гг.) наблюдается резкий скачок количества студентов, получающих образование в этой области. Мне кажется, что и падение, и скачок заслуживают объяснения.

Расхожее мнение состоит в том, что компьютерная наука переживала длительный упадок, потому что студенты считали ее скучной.[1] Я ее таковой не считаю, и меня подобные мнения не убеждают. Правильное объяснение уже много лет у нас перед глазами: количество студентов, получающих специальность по компьютерной науке, снижалось, потому что это образование все меньше увязывалось с требованиями рынка труда. Студенты знали, что можно получить работу и не имея диплома по компьютерной науке, а многие работодатели не ценили дипломы по этой специальности.

---

[1] На момент этого наблюдения программы по компьютерным наукам аккредитовались в CSAB. Теперь эти программы аккредитуются в ABET/CAC.

[1] *USA Today*, February 16, 1998, pp. 1B–2B. «Software Jobs Go Begging», *The New York Times*, January 13, 1998, p.A1.

Рис. 18.1. *Количество студентов, получивших дипломы по компьютерной науке, менялось в последние годы. Источник: Национальный центр статистики образования*[1]

Старая система образования перестала срабатывать, поэтому ее надо было изменить.

В период с 1998 по 2000 гг. начались изменения по двум направлениям. Во-первых, Интернет вызвал бурю энтузиазма среди студентов-выпускников. Началась «золотая лихорадка», и все больше студентов стало готовиться к карьере в отрасли ПО. Во-вторых, спрос интернет-компаний с названиями на «.com» в конце 1990-х годов докатился до университетской системы, и университеты снова стали увязывать программы курсов по компьютерной науке с требованиями отрасли после нескольких лет падения их популярности среди студентов.

Подготовка профессиональных инженеров

Тезис, который я отстаиваю в этой книге, состоит в том, что инженерные профессии являются отличной моделью для формирования профессии разработчика ПО, и лестница профессиональной подготовки инженеров тоже.

---

[1] Данные для построения диаграммы взяты из таблицы 255 «Bachelor's degrees conferred by degree-granting institutions, by discipline division: с 1970–1971 по 1999–2000 гг.» [97].

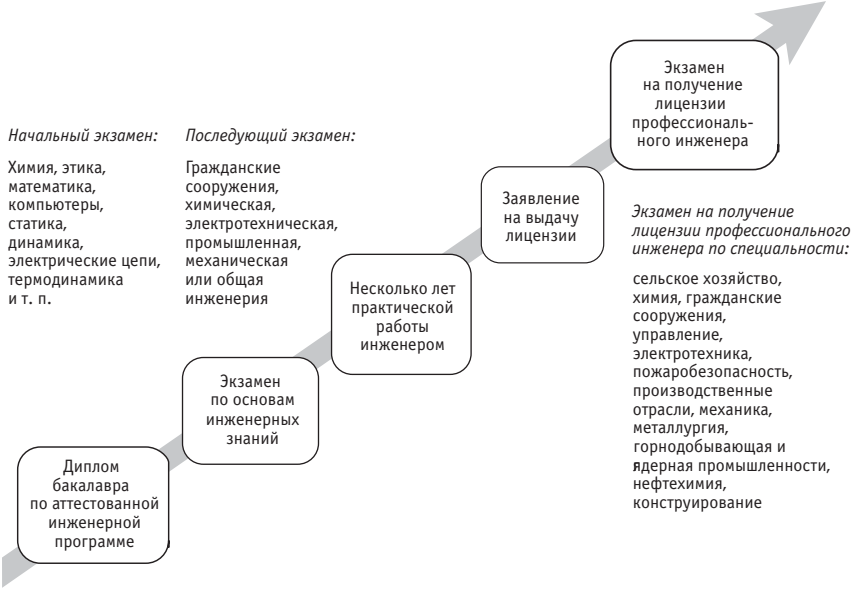

Рис. 18.2. *Профессиональная подготовка инженера включает образование, практическую стажировку и опыт работы. Это хорошая модель профессионального развития в инженерии ПО. Источник: работа Дениса Фрейли*[1]

От профессионального инженера требуются глубокие теоретические и практические знания. Как показано на рис. 18.2, профессиональный инженер сначала получает диплом бакалавра наук в аттестованном учебном заведении. Далее он сдает экзамен по основам инженерной науки (Fundamentals of Engineering – FE), который иногда называют «экзаменом инженера-адъюнкта» (EIT). После сдачи этого экзамена в течение нескольких лет новоиспеченный специалист работает под руководством профессионала. В конце этого периода инженер сдает экзамен по избранной инженерной специальности (инженер гражданских сооружений, инженер-электротехник или химик). После сдачи этого экзамена соответствующий территориальный орган выдает лицензию профессионального инженера, и появляется новоиспеченный Профессиональный Инженер.

Необходимый баланс между теорией и практикой в образовании, которое получают специалисты по инженерии ПО, достигается, если между образованием и обучением проводится четкое различие. *Образование* должно

[1] Деннис Фрейли (Dennis Frailey), частная беседа.

привить студентам качества, которые позволят им быстро справляться с разнообразными интеллектуальными задачами. Оно ориентировано на общие знания и включает развитие важнейших навыков мышления. *Обучение* дает конкретные умения и знания, которые можно применять сразу и многократно. Образование – это стратегическая часть; обучение – тактическая.

Сегодня самый распространенный тип подготовки разработчиков ПО – обучение, которое, как правило, скорее является ответом на конъюнктуру и обеспечивается в нужный момент по технологиям, которые необходимы для конкретного проекта. Образование, дающее знание основополагающих принципов инженерии ПО, почти отсутствует в этой схеме. Кое-кто утверждает, что разработка ПО стала настолько специализированной и фрагментированной, что не поддается стандартизованному образованию. Она действительно фрагментирована – для стандартизованного обучения, но не для стандартизованного образования.

Первые шаги

Программы образования для студентов-выпускников по специальности «инженерия ПО» существуют уже около 20 лет, но программы для студентов младших курсов еще только зарождаются, особенно в Северной Америке. Университет Сиэттла присвоил первую в мире степень магистра по специальности «инженерия ПО» в 1982 г. Факультет компьютерной науки в Университете Шеффилда в Англии открыл курс по данной специальности в 1988 г. Рочестерский технологический институт (R.I.T) открыл первую программу по курсу инженерии ПО в США, принял студентов в 1996 г., и первый выпуск состоялся только в 2001 г.

На сегодня в США предлагается 25 программ на получение степени магистра инженерии ПО. Несколько программ предлагается в Канаде, Великобритании, Австралии и других странах.[1] К лету 2003 г. два с половиной десятка университетов в США и Канаде предлагали дипломы низших степеней по инженерии ПО. По крайней мере 13 университетов предлагают такие программы в Великобритании и еще не менее шести в Австралии.

R.I.T. и другие университеты сотрудничают с IEEE, ACM и ABET/EAC (Аккредитационной комиссией в области техники) в разработке программ, которые могут быть аттестованы в США как инженерные программы.

---

[1] Актуальная информация на сайте моей компании по адресу *www.construx.com/profession*.

На рис. 18.3 приведен примерный список дисциплин, предусмотренных программой Рочестерского технологического института.

Программа по специальности инженерия ПО в R.I.T.

Первый год

Семинар для вновь поступивших (1)*
Компьютерная наука I, II, III (12) (основы компьютероведения)
Вычисления I, II, III (12)
Химия для колледжей I (4)
Университетский курс физики I, II и лабораторные занятия (10)
Общеобразовательные предметы (12)

Второй год

Инженерия подсистем ПО (4)
Компьютерная наука IV (4) (структуры данных)
Профессиональные взаимоотношения (4)
Введение в инженерию ПО (4)
Программирование на языке Ассемблера (4)
Введение в цифровые системы (4)
Дифференциальные уравнения (4)
Дискретная математика I, II (8)
Университетский курс физики (III) и лабораторные занятия (5)
Общеобразовательные предметы (8)

Третий, четвертый и пятый годы

Принципы архитектуры ПО (4)
Формальные методы проектирования и конструирования (4)
Технические требования и спецификации ПО (4)
Проект инженерии ПО I,II (8)
Программирование научных приложений (4)
Концепции языков программирования (4)
Компьютерные организации (4)
Человеческие факторы (4)
Теория вероятности и статистика (4)
Курс по выбору отрасли приложения** (12)
Курс по выбору (4)
Общеобразовательные предметы (18)
Курс по выбору проектирования ПО (8)
Курс по выбору проектирования ПО (8)
Курс по выбору инженерии ПО (4)
Комбинированное образование (требуется 4 квартала)

* В скобках указано количество 4-х часовых единиц (двойных «пар» академических часов).

** Каждый студент обязан пройти три курса дисциплин в области приложений, связанных с инженерией ПО. Сегодня эти области включают: электротехнику, промышленные технологии, механику, связь и сетевые технологии, встроенные системы и коммерческие приложения.

Рис. 18.3. *Программа Рочестерского технологического института по специальности «инженерия ПО» предусматривает такие дисциплины, как компьютерная наука, связь, общеобразовательные предметы и инженерия ПО*

В программу R.I.T. входит несколько дисциплин факультета компьютерной науки (например, компьютерная наука I, II, III и IV), а также несколько предметов, обычно не включаемых в программу по компьютерной науке, в том числе: инженерия подсистем ПО, требования к спецификации ПО, проект инженерии ПО I и II, а также человеческие факторы. Программа также предусматривает 4 четверти комбинированного образования (сочетание теории и практики, стажировка) и значительный опыт работы в коллективе разработчиков. Другими словами, студент должен накопить широкий опыт отраслевой практики перед получением диплома со степенью. Это требование к накопленному опыту является отличительной чертой программы инженерного образования. Программы по компьютерной науке могут предусматривать получение отраслевого опыта, но лишь за счет ослабления упора на чистую науку.

Одна интересная особенность программы R.I.T. состоит в ее продолжительности (5 лет). В середине 90-х годов XX века степень бакалавра в инженерной науке можно было получить, как правило, после пяти лет обучения [46]. Различные факторы вынудили университеты сократить продолжительность обучения до четырех лет. Четырехгодичное обучение инженерии по программе низшей степени, наверное, неразумно с практической точки зрения, особенно по модели R.I.T., где студенты целый год приобретают отраслевой опыт. Программы, рассчитанные на пять лет, могут стать нормой для получения первой степени по специальности «инженерия ПО».

Аттестация

Аттестация университетских программ требуется для поддержания высоких стандартов образования в инженерии ПО. Она призвана гарантировать, что студенты, прошедшие обучение по аттестованной программе, получат основные знания в выбранной области. Аттестация также обеспечивает общую рабочую терминологию и стандарты выполнения работы. В США программы инженерного образования аттестует ABET/EAC, которая не выдает сертификата, пока не сделан первый выпуск специалистов. Как только будет аттестована первая программа, многие другие университеты, вероятно, почти сразу же станут предлагать свои программы по инженерии ПО на получение начальных степеней.

Одно из различий компьютерной науки и инженерии ПО заключается в аттестационных требованиях к преподавателям по данным специальностям.

Критерии аттестации программ по компьютерной науке в США требуют от профессорско-преподавательского состава вносить в нее «научно-исследовательский вклад», но не требуют иметь отраслевой опыт [31]. В отличие от этого, критерии ABET/EAC для аттестации программ инженерного образования в США устанавливают, что при оценке профессорско-преподавательского состава необходимо учитывать «непреподавательский инженерный опыт», а регистрацию в качестве профессиональных инженеров [32]. CEAB (Канадский совет инженерной аттестации) применяет аналогичные критерии. В профессиональном образовании важно, чтобы многие преподаватели были квалифицированными действующими специалистами по профессии – врачи учат врачей, адвокаты – адвокатов и т. д.

Эти различия между критериями аттестации программ по инженерии и компьютерной науке не означают, что один подход правильный, а дру-

гой нет. Просто у них разные цели. Научные программы готовят студентов к исследовательской работе, а инженерные – к работе в отрасли. Отрасль ПО остро нуждается в специалистах, хорошо подготовленных для работы именно в ней. Программы для получения первых степеней в инженерии ПО показывают свою популярность как среди студентов, так и среди компаний, которые их нанимают.

Конструирующие программисты или программирующие инженеры?

Еще один вопрос, который требует ответа: на чем следует делать больший акцент – на применении общих инженерных подходов к разработке ПО или на роли прикладной инженерной специфики в разработке программного обеспечения?

Согласно одному из направлений, представленному программой на степень бакалавра в R.I.T., инженер ПО должен уметь разрабатывать ПО, применяя принципы и подходы, *характерные* для инженерных дисциплин в целом, и знакомство с традиционно инженерными предметами для него не обязательно. В этом случае потенциальный инженер ПО будет изучать университетские курсы математики, компьютерной науки, управления и курсы по специальности ПО, но не будет проходить *все* предметы, традиционно связанные с инженерией.

Согласно второму направлению инженер ПО – это инженер в традиционном понимании, получивший специальное образование в области разработки ПО. Дэвид Парнас из Университета Мак-Мастера в провинции Онтарио (Канада) соглашается, что для получения профессионального мандата и уважения других инженеров инженеры ПО должны иметь такое же фундаментальное образование, что и другие профессиональные инженеры. Инженеры ПО часто занимаются авиационным ПО, контролем производства, работают на атомных станциях и в других сферах, где знание инженерных основ полезно или необходимо. Студенты, получающие подготовку в этих областях инженерии ПО, проходят университетские курсы общей химии, прикладной математики, инженерных материалов, термодинамики и теплопроводности, а также курсы других дисциплин, являющихся частью традиционного базового образования инженера. Дополнительно в их обучение включены курсы дисциплин, являющихся частью традиционного образования для получения диплома по компьютерной науке.

В табл. 18.1 приведена сводка программ образования в Университете Мак-Мастера и Рочестерском технологическом институте.[1]

Таблица 18.1. Требования к курсам по основным дисциплинам для двух типов инженерии ПО

	РТИ (R.I.T.)	Университет Мак-Мастера
Математика и научные дисциплины Химия, анализ, теория матриц, комплексные числа, дифференциальные уравнения, дискретная математика, теория вероятности и статистика.	✓	✓
Введение в инженерные дисциплины Структура и свойства инженерных материалов, динамика и контроль физических систем, термодинамика и теплообмен, волны, электричество и магнитные поля.	–	✓
Компьютерная наука Введение в программирование, принципы цифровых систем, архитектура компьютеров, логическое проектирование, алгоритмы и структуры данных, программирование на машинном языке, концепции языков программирования, методы оптимизации, графические модели, алгоритмы поиска и отсечения тупиковых ветвей.	✓	✓
Разработка программного обеспечения Архитектура и проектирование ПО, формулировка технических требований, навыки профессионального общения; проектирование ПО разделения времени и ПО реального времени, проектирование параллельных и распределенных систем; вычислительные методы в науке и инженерии, проектирование интерфейса пользователя.	✓	✓
Управление разработкой ПО Метрики процесса разработки ПО и программного продукта.	✓	–
Информационно-управляющие системы (MIS) Принципы проектирования информационных систем.	✓	–

[1] Программа обучения Университета Мак-Макстера имеется на сайте *www.cas.mcmaster.ca/cas/undergraduate*. Программа института R.I.T. размещена по адресу *www.se.rit.edu*.

Разница в программах Университета Мак-Мастера и R.I.T. указывает на важные различия в философии понимания инженерии ПО. Программа Университета Мак-Мастера считает специалистов ПО инженерами, разрабатывающими ПО. Программа R.I.T. рассматривает инженеров ПО как программистов, создающих ПО на основе инженерного подхода.

Инженеры ПО, получившие образование по программе, подобной принятой в Университете Мак-Мастера, смогут сдать экзамен по основам инженерии, который должны сдавать все профессиональные инженеры в США и Канаде, и получить в итоге лицензию профессионального инженера (P.E.) в США или ее эквивалент (P.Eng.) в Канаде. Программы, подобные предлагаемой Рочестерским технологическим Институтом, не дадут инженерам ПО основ инженерии, необходимых для успешной сдачи экзамена по соответствующей дисциплине.

Ценность каждой из программ несомненна. Более распространена в отрасли ПО необходимость в «*конструирующих программистах*», которые создают практичные и экономичные бизнес-системы, однако уровень критичности некоторых систем (например, ПО ядерных электростанций и самолетов) в равной степени требует «*программирующих инженеров*». Некоторые эксперты считают, что есть только один настоящий тип инженерии ПО, но мне представляется правильным называть оба типа программ образования «инженерия ПО», а выпускников, прошедших полный курс по соответствующим программам, именовать инженерами ПО.

Дифференцированность этих подходов устанавливает различные связи с существующими инженерными дисциплинами, лицензированием и сертификацией программ. Мы вернемся к этому вопросу в главе 19.

Полировка жетона

К профессионалу, получившему базовое образование, некоторый опыт и, возможно, лицензию, в большинстве профессий предъявляется требование непрерывного обучения. Конкретные требования в разных штатах разные. В штате Вашингтон от сертифицированных бухгалтеров требуется получить 80 зачетных единиц непрерывного профессионального образования (Continuous Professional Education, CPE) за два года, предшествующие продлению их квалификационных сертификатов.[1] Ад-

[1] Интернет-представительство вашингтонского общества сертифицированных бухгалтеров: *www.wscpa.org*.

вокаты должны каждый год набирать 15 зачетных единиц непрерывного юридического образования (CLE). Врачи в штате Нью-Мексико должны получать 150 часов непрерывного образования каждые три года. К инженерам в штате Вашингтон не предъявляется никаких требований по непрерывному образованию, а в других штатах они существуют [29].

Непрерывное образование позволяет обеспечить актуальность знаний специалистов в соответствующих областях, что особенно важно, например, для медицины или инженерии ПО, где знания постоянно обновляются. Если специалист, получив базовое образование, прекращает учиться, его профессионализм со временем начинает снижаться.

Непрерывность образования стимулируется требованием к специалистам быть в курсе важных событий, новшеств в их области деятельности. Если в инженерии ПО когда-либо сыщется новое универсальное средство типа «серебряной пули», то требование непрерывного образования обеспечит знакомство с ним всех лицензированных или сертифицированных инженеров.

Некоторые перспективы

Инженерии ПО всего каких-то 50 лет. За это время компьютерные программы изменили мир до такой степени, что нам трудно представить его без ПО. Как и в других инженерных дисциплинах, практика большей частью подстегивала теорию, и университетское образование не всегда за ними успевало. С другой стороны, некоторые теоретические разработки ставили практиков в положение догоняющих, потому что общий уровень образования в инженерии ПО был весьма недостаточен. Не имея университетской инфраструктуры, трудно воплощать в жизнь проверенные теории.

Образовательные программы в области инженерии ПО, созданные по образу и подобию традиционных программ инженерного образования (создающие «программирующего инженера»), – это как раз то, что нужно. Они подготовят выпускников, которые принесут отрасли больше пользы, чем специалисты по компьютерной науке. Это позволит программам образования по компьютерной науке избавиться от существующего раздвоения и больше сосредоточиться на науке. И гораздо легче будет получить образование в области инженерии ПО всем заинтересованным в нем, чем набивать шишки в школе жизни.

ГЛАВА ДЕВЯТНАДЦАТАЯ

Кому нужны дипломы?

Жетоны? Нет у нас никаких жетонов. Не нужны нам жетоны. Не должен я показывать какие-то вонючие жетоны.

Бандито Золотая кепка «Сокровища Сьерра Мадре»

Не многие вопросы вызывают столь ожесточенные споры, как проблема сертификации и лицензирования. Но проку от этих споров намного меньше, чем шума. Царит вопиющее непонимание разницы между лицензированием и сертификацией. Обязательна ли сертификация? Нужно ли получать лицензию? Как это затрагивает специалистов? Этим вопросам и посвящена данная глава.

Сертификация

Сертификация – это *заявительная* процедура, контролируемая профессиональным сообществом, цель которой – информировать общество о том, что квалификация заявителя достаточна для выполнения определенной работы (или о соответствии товара, услуги требуемым стандартам). Для сертификации обычно требуются и образование, и практический опыт. В большинстве случаев компетентность претендента на сертификацию определяется по результатам письменного экзамена. Действие сертификации, как правило, распространяется за пределы ограниченной географической зоны, захватывая национальный или международный уровень. Самый известный пример сертификации в США – дипломированный бухгалтер (Certified Public Accountant, CPA).

Некоторые организации уже много лет предлагают сертификацию специалистов в области ПО. Институт сертификации профессионалов в облас-

ти использования компьютеров присваивает звания кандидата в профессионалы и сертифицированного профессионала. Американское общество контроля качества присваивает звание инженера по качеству ПО (хотя употребление термина «инженер» подвергает это общество юридическим рискам, поскольку его применение регулируется в большинстве штатов США [69] и в Канаде).

Многие компании имеют программы сертификации по собственным конкретным технологиям. Microsoft предлагает диплом сертифицированного специалиста Microsoft; Novell выдает диплом сертифицированного инженера сетей, у Oracle есть сертифицированные специалисты Oracle, а Apple Computers присваивает звание профессионального инженера серверов Apple. Сфера такой сертификации ограничена продуктами одной компании, что делает ее скорее сертификацией в технологии, чем в инженерии ПО.

Сертификация дает возможность работодателям и клиентам получить информацию о специалистах в сфере ПО, имеющих по крайней мере какую-то минимальную квалификацию. И рынок уже отреагировал на этот запрос. Во время написания этой книги на сайте Amazon.com размещалось 25 *категорий* книг по различным сертификационным экзаменам в области ПО и связанных с компьютерами дисциплин. Почти все эти экзамены относились к конкретным технологиям.

В 2002 г. компьютерное общество IEEE стало проводить сертификацию по основам инженерии ПО на звание сертифицированного специалиста разработки ПО[1]. Это первая общая сертификация в области инженерии ПО, предлагаемая крупной профессиональной организацией, и это существенный шаг к признаваемой отраслью системе аттестации профессионалов разработки ПО.

Лицензирование

Лицензирование – это *обязательная* разрешительная юридическая процедура, призванная защитить интересы общества и обычно осуществляемая территориальными органами юрисдикции (штатов, провинций и т. д.). Во многих профессиях национальные организации оказывают территориальным органам помощь в разработке лицензионных требований и формировании программ экзаменов.

[1] Подробности см. на сайте *www.computer.org/certification*.

Большинство профессий лицензируется (например, врачи, архитекторы, юристы и инженеры). Ни одна из профессий, столь же сильно влияющих на жизнь общества, как ПО, не осталась нелицензируемой. В табл. 19.1 приведен перечень некоторых профессий, лицензируемых в штате Калифорния.

Таблица 19.1. Примеры родов деятельности, требующих получения лицензии в Калифорнии [46]

- Иглотерапевт
- Оператор охранной компании
- Боксер-любитель
- Архитектор
- Адвокат
- Парикмахер
- Дипломированный бухгалтер
- Строитель-подрядчик
- Косметолог
- Обивщик мебели
- Дантист
- Семейный консультант
- Похоронный распорядитель
- Геолог
- Инструктор собак-поводырей
- Продавец слуховых аппаратов
- Жокей
- Слесарь по замкам
- Мастер маникюра
- Наездник на мулах
- Няня
- Оператор контроля за насекомыми
- Врач
- Ассистент врача
- Частный сыщик
- Профессиональный инженер
- Оценщик недвижимости
- Розничный торговец мебелью
- Ветеринар

В инженерной сфере США большинству специалистов не требуется получать лицензию. Инженерные фирмы должны иметь некоторое количество лицензированных инженеров, но не все инженеры обязаны иметь лицензии. Так, лицензируется половина инженеров-строителей, а среди инженеров-химиков – лишь 8%. Различие заключается в степени повторяемости проектируемого объекта и его влиянии на безопасность общества. Предметы, воспроизводимые в больших количествах, могут испытываться до их запуска в производство для продажи населению, что минимизирует риск для общества и позволяет уменьшить количество лицензированных инженеров, требуемых для выполнения такой работы. Инженеры-электротехники разрабатывают продукты, воспроизводимые в огромных количествах: тостеры, телевизоры, телефоны и т. д. Поэтому лишь небольшая их часть должна иметь лицензию, что видно из табл. 19.2.

Таблица 19.2. Процент лицензированных выпускников инженерных специальностей в США в 1996 г. [46]

Дисциплина	Лицензировано (%)
Гражданское строительство	44
Механика	23
Электротехника	9
Химия	8
Все инженерные специальности	2

Инженеры по гражданским сооружениям проектируют множество уникальных объектов, безопасность которых критически важна: шоссе, мосты, стадионы, взлетно-посадочные полосы и т. д. Поэтому, как показано в табл. 19.2, инженеров гражданского строительства лицензировано значительно больше, чем инженеров-электротехников.

В какую строку этой таблицы попала отрасль ПО? Разработчики ПО создают много уникальных продуктов, но также и множество продуктов широкого пользования: операционные системы, ПО заполнения налоговых деклараций, текстовые процессоры и другие программы, которые воспроизводятся в миллионах копий. Критически важные для безопасности общества системы также разрабатываются, но значительно больше систем, меньше влияющих на нее, создается для бизнеса.

Перестраивая здание, строитель может самостоятельно принимать многие проектные решения. Если решение может повлиять на прочность здания, то для оценки такого решения строителю требуется помощь инженера. В отрасли ПО даже приложения, требующие инженерного подхода, могут большей частью создаваться «строителями» – нелицензированными инженерами ПО и специалистами по технологиям. Поэтому большинство приложений ПО не обеспечено инженерией вовсе, а там, где требуется какая-то часть инженерии, приложения могут создаваться персоналом, лишь малая доля которого является лицензированными инженерами ПО.

Когда ситуация стабилизируется, по моим подсчетам, менее чем 5% практикующих сегодня программистов в конечном итоге потребуется получить лицензию инженера ПО, и эта доля может легко приблизиться к одному проценту.[1]

[1] Форд и Гиббс делают аналогичное замечание – по их оценкам, менее 10% разработчиков ПО захотят получить лицензии [46].

Возможно ли лицензирование инженеров ПО

Некоторые эксперты компьютерной науки утверждают, что лицензирование будет неэффективным или даже контрпродуктивным [38], [70], [71], [82], [141]. Эти заявления делаются слишком часто, чтобы оставить их без рассмотрения.

Суть некоторых из них сводится к тому, что лицензирование в инженерии ПО невозможно или практически неразумно, а других – к тому, что идея лицензирования плоха сама по себе.

Вот краткий перечень аргументов в пользу непрактичности или невозможности лицензирования в ПО:

- Для инженерии ПО не определен общепринятый объем знаний [71].
- Знания в инженерии ПО меняются так быстро, что экзаменационные материалы устареют к моменту экзамена [71], [82].
- Трудно втиснуть сколько-нибудь осмысленный тест по инженерии ПО в формат выбора ответа из нескольких вариантов. Фактически никакая методика, основанная на экзаменах, не сможет адекватно удостоверить компетентность инженера ПО [71].
- Спектр поддисциплин, вовлекаемых в разработку ПО, настолько широк, что попытки лицензировать их все не имеют практического смысла [71], [82].
- Экзамен по основам инженерии для лицензирования работающих инженеров не годится для тех, кто получил образование по специальности «компьютерная наука» [71], [141].

Рассмотрим каждый из этих аргументов.

Для инженерии ПО не определен общепринятый объем знаний.

Если лет тридцать назад это утверждение, возможно, и было похоже на правду, то сегодня оно неверно. Как говорилось в главе 5, предметный объем знаний инженерии ПО хорошо определен и достаточно стабилен.

Знания в инженерии ПО меняются так быстро, что экзаменационные материалы устареют к моменту экзамена.

Это утверждение также основано на устаревшем понимании предметной области знаний инженерии ПО, описанной в главе 5.

Знания в некоторых других сферах (особенно в медицине) меняются, по крайней мере, так же быстро, как и в ПО. Если возможно лицензировать врачей, можно лицензировать и инженеров ПО.

Трудно втиснуть сколько-нибудь осмысленный тест по инженерии ПО в формат выбора ответа из нескольких вариантов. Фактически никакая методика, основанная на экзаменах, не сможет адекватно удостоверить компетентность инженера ПО.

Формирование экзамена профессионального уровня действительно серьезная проблема. Однако наука и методика разработки экзаменов на профессиональную пригодность достаточно хорошо развиты. На экзаменах основано лицензирование во многих профессиях: от медицины и права до теории страхования и других дисциплин. Разработка экзамена требует много времени, сил, а также привлечения настоящих знатоков своего дела. Разработка экзамена для сертифицированного специалиста по разработке ПО, как я испытал на собственном опыте,[1] – это серьезная задача, но не более, чем в любой другой профессии.

Спектр поддисциплин, вовлекаемых в разработку ПО, настолько широк, что попытки лицензировать их все не имеют практического смысла.

Разнообразие стилей разработки ПО действительно представляет собой серьезную трудность для лицензирования инженеров ПО. К счастью, дело упрощается, поскольку, как и в других дисциплинах, большинству инженеров нет нужды получать лицензию. Лицензировать надо лишь тех, кто разрабатывает ПО, представляющее риск для здоровья или благосостояния населения.

Вопрос о «широте поддисциплин» не уникален. Врачи сдают отдельные экзамены по кардиологии, радиологии, онкологии и другим специальностям. Инженеры сдают специализированные экзамены по гражданскому строительству, электротехнике, химии и т. д. Если в других сферах это препятствие смогли преодолеть, то оно будет преодолено и в ПО.

Взвесив все эти соображения, я считаю, этот аргумент скорее говорит не против лицензирования, а в пользу реалистичных ожиданий результатов, которых можно достичь лицензированием. В других инженерных дисциплинах действует кодекс поведения, согласно которому специалисты не должны практиковать вне сферы своей компетенции. Инженерия ПО нуждается в аналогичном стандарте.

Экзамен по основам инженерии для лицензирования работающих инженеров не годится для тех, кто получил образование по специальности «компьютерная наука».

---

[1] См. *www.computer.org/certification.*

Экзамен по инженерным основам сегодня сосредоточен на основных положениях инженерии, включая структуру и свойства инженерных материалов, динамику и управление физическими системами, термодинамику и теплопроводность, магнетизм и т. д. Как говорилось в главе 17, некоторые формирующиеся учебные программы инженерии ПО действительно требуют от студентов изучения этих традиционных инженерных предметов. Другие программы больше сконцентрированы на ПО и меньше на традиционных дисциплинах, поэтому студенты, выпущенные по таким программам, не подготовлены для экзамена по инженерным основам.

Не вызывает сомнений, что предстоит выполнить огромную работу, прежде чем лицензирование в инженерии ПО станет реальностью. Но сейчас эта работа во многом выполнена, поскольку она проводится уже несколько лет. С моей точки зрения, все споры о практической пользе лицензирования имеют лишь косвенное значение; суть вопроса в том, правильна ли сама идея лицензирования.

Правильна ли сама идея лицензирования?

Вот список аргументов в пользу того, что лицензирование – это плохая идея:

- Лицензии искусственно сократят количество работников, которые могли бы заниматься инженерией ПО, когда спрос на инженеров ПО растет [70].
- Инженер получает бессрочную лицензию, что неразумно в условиях быстро меняющегося объема знаний инженерии ПО [82].
- Лицензирование не сможет гарантировать компетентность каждого обладателя лицензии. Лицензирование даст обществу ложное ощущение защищенности [71].

Разберем теперь каждый из этих аргументов.

Лицензии искусственно сократят количество работников, которые могли бы заниматься инженерией ПО, когда спрос на инженеров ПО растет.

Это утверждение основано на предположении, что после внедрения лицензирования лишь имеющие лицензию инженеры ПО смогут его создавать. Как уже отмечалось в данной главе, в других сферах лицензирование работает не так, не по этому правилу оно будет работать и в ПО. Большинство работников, пишущих программы, будут специалистами по тех-

нологиям или нелицензированными инженерами ПО, а не профессиональными инженерами. Большинство типов ПО не несет в себе риска и может быть написано непрофессиональными инженерами ПО.

Лишь немногим инженерам ПО, создающим определенные типы систем ПО, критически важных для безопасности людей, вообще потребуется иметь лицензию.

Инженер получает бессрочную лицензию, что неразумно в условиях быстро меняющегося объема знаний инженерии ПО.

Это утверждение основано на неправильном понимании двух аспектов лицензирования в инженерии ПО: во-первых, быстроты изменений предметной области знаний (глава 5) и, во-вторых, предположения, что лицензия выдается бессрочно. Одно из условий лицензирования вообще состоит в непрерывности профессионального образования, дающего возможность оставаться в курсе всех изменений (в том числе и чтобы сохранить лицензию). Лицензирование *стимулирует* актуальность знаний профессионалов.

Лицензирование не сможет гарантировать компетентность каждого обладателя лицензии. Лицензирование даст обществу ложное ощущение защищенности.

Утверждение о том, что кто-то из неквалифицированных претендентов может заполучить лицензию, тогда как в ней будет отказано более квалифицированным, отчасти справедливо. Лицензирование действует как фильтр, отсеивающий худшее и повышающий уровень квалификации работников, однако этот механизм не абсолютно безупречен. На рис. 19.1 показан контингент работающих в отсутствие лицензирования.

Без профессионального лицензирования дорога будет открыта не только надежным методикам разработки ПО, но и негодным, и даже потенциально опасным. Чтобы защитить интересы общества, механизм лицензирования должен работать как фильтр, отсеивая худших и выдавая лицензии только лучшим разработчикам ПО (рис. 19.2).

Будет правильно предположить, что лицензирование не станет идеальным фильтром. Все мы слышали про хороших и плохих врачей, адвокатов и представителей других профессий. Даже в сочетании с университетским дипломом, экзаменами и практическим опытом лицензирование в ПО не будет лучше, чем в существующих профессиях. Поначалу оно скорее будет даже хуже, потому что в других профессиях экзамены и другие требования лицензирования корректировались и уточнялись. Как показано на рис. 19.3,

Рис. 19.1. *Контингент всех разработчиков ПО до применения лицензионного фильтра*

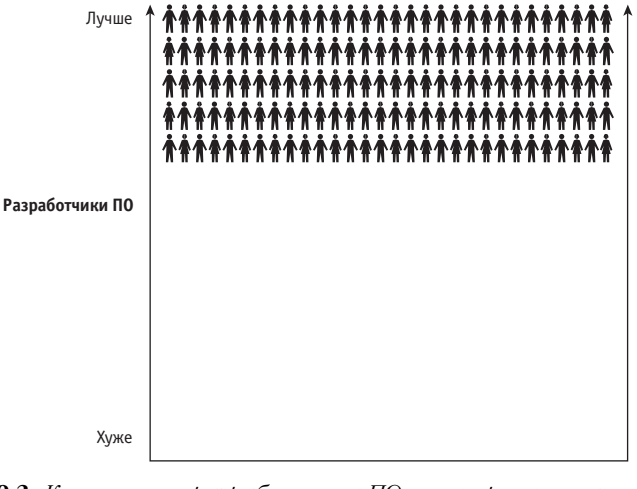

Рис. 19.2. *Контингент разработчиков ПО после применения лицензионного фильтра (идеальная фильтрация)*

даже лучшие современные подходы к лицензированию в ПО могут оставить лазейку для неквалифицированных работников, позволяя им проникать в отрасль, и отсечь тех, кому действительно надо было выдать лицензию.

Идеальное лицензирование, наверное, недостижимо, но лицензирование реальное все же имеет свою ценность. Большинство работодателей

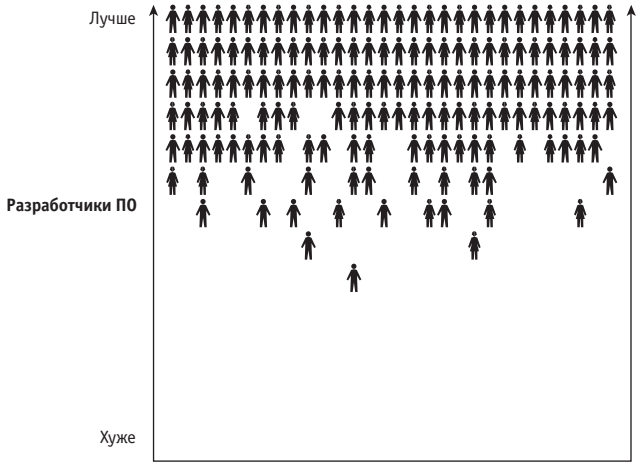

Рис. 19.3. *Контингент разработчиков ПО после применения лицензионного фильтра (реальная фильтрация)*

в индустрии ПО скорее выберут себе работников из контингента на рис. 19.3, и не прельстятся вариантом, показанным на рис. 19.1. Большинство населения предпочтет, чтобы критически важное для безопасности ПО разрабатывалось и проверялось специалистами из контингента на рис. 19.3. Гарантии хороши тогда, когда они есть. Когда их нет, то лучше большая уверенность, чем меньшая.

Раскрутка лицензирования

Движение за лицензирование разработчиков ПО по-настоящему началось в 1998 г., когда Совет профессиональных инженеров Техаса принял инженерию ПО как отдельно лицензируемую дисциплину с присвоением звания профессионального инженера (или P.E.) специалистам в области ПО.[1]

Вместо общеиспользуемого экзамена в Техасе начали лицензировать инженеров-разработчиков ПО согласно «правилу допуска к экзамену». Чтобы получить лицензию P.E., прежде чем приступить к экзамену, претендент должен был предъявить:

- 16 лет практического инженерного опыта или

[1] *www.tbpe.state.tx.us.*

- 12 лет инженерной практики и диплом бакалавра по аттестованной университетской программе.
- Помимо этого каждый претендент должен был представить по крайней мере 9 рекомендаций, не менее пяти из которых должны были дать лицензированные инженеры (необязательно в области разработки ПО).

Те же самые критерии допуска к экзамену применяются в Техасе и к другим инженерным специальностям.

Сколько практикующих разработчиков ПО смогли получить лицензию профессионального инженера ПО согласно установленной процедуре лицензирования? Около 50 было лицензировано в середине 2003 г., что не слишком много; и это одна из самых разумных вещей, сделанных в Техасе. Естественным было бы желание ослабить строгость отбора на начальном этапе лицензирования, чтобы большинство действующих инженеров автоматически подпадали под требования. Итогом этого стала бы девальвация звания «профессиональный инженер ПО» до уровня, соответствующего обычному (начинающему) программисту. Ужесточив правила, в Техасе максимально увеличили вероятность того, что инженеры ПО в штате будут принадлежать к когорте лучших разработчиков ПО, и защитили их репутацию.

Аналогичные события имели место и в Канаде. Провинции Британская Колумбия и Онтарио начали выдавать лицензии профессиональной инженерии ПО в 1999 г., и в Онтарио лицензии получили около 300 профессионалов [109]. Как и в Техасе, программы лицензирования в обеих провинциях предусматривали дополнительные условия, действующие на начальном этапе, позволяющие разработчикам ПО, имеющим соответствующее образование и практический опыт, получить лицензии инженеров ПО, пока подготавливались экзамены.[1]

Ваша ставка

Одно из следствий обладания лицензией профессионального инженера состоит в личной ответственности за работы, выполняемые для

[1] То есть во всех описываемых случаях действует принцип: для получения лицензии надо успешно пройти некоторую систему тестов, но для допуска к процедуре прохождения требуется предварительно предоставить свидетельства своего практического опыта и профессионализма. – *Примеч. науч. ред.*

компании, то есть за авторство. Суды в США постановили, что только члены профессиональных сообществ могут признаваться виновными в преступной халатности [69]. Врачи, адвокаты и архитекторы[1] могут быть признаны виновными. Водители мусоросборщиков, повара и компьютерные программисты не могут, потому что юридически не считаются профессионалами. Закрепив за инженерией ПО статус профессии, мы откроем дорогу для судебного преследования инженеров ПО, обвиняемых в преступной халатности так же, как других профессионалов.

Отдельный инженер не обязан получать лицензию, но от некоторых компаний потребуется иметь в штате лицензированных инженеров. Вероятнее всего, что это будут компании:

- поставляющие услуги инженерии ПО населению;
- выполняющие работы ПО для государственных агентств;
- производящие ПО, критически важное для безопасности людей.

Другие компании могут по собственному желанию привлекать профессиональных инженеров, чтобы воспользоваться преимуществом «блестящей упаковки», которое обеспечивается репутацией лучших специалистов отрасли, или потому что видят в этом средство укрепления своего инженерного персонала. Наличие в штате разработчиков ПО, которые получили сертификаты, но не профессиональный статус в инженерии, может служить интересам компаний столь же хорошо.

Профессиональные инженеры в таких компаниях будут производить ревизию разработки и ставить «знак качества» на выпускаемое ПО. Привлечение лицензированных инженеров станет жизненно необходимым для компаний. Как и в других инженерных отраслях, компания, нанимающая профессионального инженера, оплачивает страховку его личной ответственности, что сводит к минимуму издержки процесса получения требуемой лицензии.

Профессиональные инженеры получат другие преимущества. Те из них, кто ставит свою подпись и репутацию под угрозу риска ради своей компании, получат от нее в конечном итоге право выбора методик, подходов к проектированию и решений, влияющих на качество ПО, за которые им придется отвечать. Не имея статуса профессионала, инженеру ПО трудно будет противостоять руководству, которое может настоять на принятии нереальных обязательств по срокам, потребовать закрыть глаза на изъяны проекта или пожертвовать качеством ради скорейшего выпуска ПО. Если

[1] Лицензированные. – *Примеч. науч. ред.*

профессия сформируется со всеми своими атрибутами – образованием, кодексом профессиональной этики и лицензированием, то появится возможность сказать: «Стандарты профессии не позволяют в данной ситуации пренебречь качеством. Можно потерять лицензию или быть привлеченным к судебной ответственности». Создаются юридическая и профессиональная основа для сопротивления недальновидным менеджерам, маркетологам и клиентам, которая начисто отсутствует сегодня.

На организационном уровне можно будет наблюдать взаимную корреляцию между рейтингом (компании) по SW-CMM (который обсуждается в главе 14) и лицензированием инженерии ПО. Профессиональные инженеры потенциально могут быть привлечены к судебной ответственности за ПО, написанное под их руководством. Они не могут лично проверить каждую строку программы в больших проектах. Даже если организации оплачивают полис страхования личной ответственности профессиональных инженеров, я думаю, что профессиональные инженеры предпочтут работать в тех организациях, где им будет обеспечена техническая и методическая поддержка, то есть где имеется наиболее развитая организационная инфраструктура разработки ПО. Профессиональные инженеры скорее всего сосредоточатся в организациях с высшими рейтингами по SW-CMM. Это укрепит тенденцию, подмеченную Харланом Миллсом 20 лет назад: разработчики выше среднего уровня собираются в эффективных организациях, разработчики уровня ниже среднего – в неэффективных [91].

Как заслужить диплом

Ключевым элементом любого зрелого плана лицензирования является экзамен на профессиональную пригодность, проводимый уполномоченным органом; однако его форма для инженера ПО пока не определена. В других инженерных областях специалисты обычно сдают восьмичасовой экзамен, включающий решение восьми задач, четыре из которых требуют ответа в описательной форме, а в четырех надо выбрать правильный ответ из примерно десяти вариантов. Конкретные особенности могут меняться в различных территориальных образованиях.

Сами по себе экзамены не дают стопроцентной гарантии, поэтому успешная сдача экзамена инженера ПО недостаточна для получения лицензии. Лицензия профессионального инженера традиционно требует как опыта работы, так и диплома со степенью аттестованного учебного заведения для инженеров. В инженерии ПО степень проблематична, поскольку

ку ни одна из программ обучения инженерии ПО, предлагаемых примерно десятью университетами, не аттестована в США, а в Канаде аттестованы три программы [109]. Аттестация первой программы обучения в США ожидается в 2003 г. сразу после выхода из печати этой книги. Можно ожидать наступления переходного периода (примерно от 10 до 15 лет) в лицензировании, когда степень не будет обязательным требованием, пока не сформируется инфраструктура университетов, обеспечивающая выпуск достаточного количества инженеров ПО.

Три пути

Как уже говорилось в главе 18, сообщество разработчиков ПО пока не пришло к единому мнению о том, кем считать специалиста ПО – инженером, разрабатывающим ПО, или программистом, который создает ПО, применяя инженерные подходы. Различие между этими двумя направлениями в понимании инженерии ПО заставляет предположить, что лицензирование в конечном итоге будет развиваться по одному из трех путей.

Первый путь подразумевает разработку специального экзамена по ПО в рамках традиционного лицензирования инженерии. Инженеры, сдавшие экзамен по основам инженерии и обладающие требуемым практическим опытом, могут получить лицензию P.E./P.Eng. в ПО, сдав экзамен по специальности «программное обеспечение». Этот путь потребует от инженеров ПО получения образования по программам типа существующей в Университете Мак-Мастера и описанной в главе 18.

Второй путь тоже предполагает разработку экзамена по специальности ПО, но требует еще и модификации экзаменов по основам инженерии, которые сдают все инженеры (конкретное содержание экзаменов меняется в зависимости от территориальной юрисдикции). Сегодня большинство инженеров активно пользуются компьютерами, а многие и создают программы для себя или для своих коллег. Эти программы применяются для формирования данных, необходимых при проектировании мостов, зданий, нефтеперерабатывающих заводов и многих сооружений, которые потенциально воздействуют на общественное благосостояние. Инженеры, пишущие такие программы, должны знать эффективные методики инженерии ПО. Возможно, потребуется просто пересмотреть содержание экзаменов по основам инженерии, увеличив в них долю вопросов по инженерии ПО. Экзамены охватывают довольно широкий круг проблем, и для их сдачи, как правило, достаточно набрать около 70% возможных баллов,

хотя высота барьера зависит от территориальной юрисдикции. Инженеры, прошедшие подготовку по другим дисциплинам, дадут больше неверных ответов на вопросы по инженерии ПО, но правильно ответят на большее количество вопросов по традиционной инженерии. Инженеры ПО правильно ответят на большее число вопросов по ПО, но дадут больше неверных ответов на традиционные вопросы по инженерии. Этот путь выбрали бы инженеры, прошедшие подготовку по программе типа предлагаемой в R.I.T., которая тоже описана в главе 18.

Третий путь предполагает создание профессионального статуса, который более тесно увязан с ПО, чем статус профессионального инженера, например «профессиональный инженер ПО – P.S.E.» или что-нибудь в этом роде. Получение этого звания было бы ориентировано исключительно на ПО и не требовало бы образования в сфере термодинамики или теплопроводности, знания материаловедения или других предметов традиционной инженерии. Один из аргументов в пользу такого подхода состоит в том, что специалистам, проектирующим бизнес-системы, финансовые приложения, образовательные программы и другое ПО, не используемое в инженерных целях, не нужно знание традиционных инженерных дисциплин, но полезно владеть инженерным подходом к созданию ПО.

Из этих трех возможных путей я считаю наилучшим второй. Инженеры ПО должны изучить достаточный спектр чисто инженерных предметов, чтобы понять образ мысли инженеров в традиционных отраслях при проектировании и решении задач, а третий путь этого вообще не требует. Но инженерам не надо изучать все инженерные дисциплины, чтобы постичь инженерный склад ума, как этого требует первый путь. Более того, в таком случае совсем мало инженеров ПО вообще получат лицензии, а программа обучения инженерии ПО должна учитывать реалии: большинству дипломированных инженеров ПО не надо готовиться к экзамену по основам инженерии. Тех же инженеров ПО, которые решат получить лицензию, второй путь подготовил бы к сдаче пересмотренного экзамена по основам инженерии, не разбавляя сверх меры специфическую для инженерии ПО сумму знаний.

Будет ли полезным пересмотр экзамена по основам инженерии для других инженерных отраслей? Думается, да. Изменение экзаменов по основам инженерии не снизит статус инженеров, которые в итоге получат лицензии. Они по-прежнему должны будут сдавать экзамены по избранной инженерной специальности по окончании своего практического обучения: по гражданскому строительству, аэрокосмической, химической ин-

женерии и т. д. Этический кодекс запрещает им практиковать в других инженерных отраслях, где они не являются специалистами. Помимо этого ПО настолько глубоко проникло в современную инженерию, что знакомство *всех* инженеров с инженерией ПО может стать существенным преимуществом: это позволит им понимать проблемы, связанные с созданием сложных систем ПО, и осознавать, когда необходимо прибегнуть к экспертной помощи в проектах, выходящих за пределы их образования и обучения.

Вонючие дипломы или стальное колечко?

В Канаде инженеры, получающие дипломы по окончании обучения по аттестованным инженерным программам, при выпуске получают стальное кольцо. С 1923 г. это кольцо вручается на закрытой церемонии, которую разработал Редьярд Киплинг. Традиция гласит, что эти колечки делаются из стали моста, который рухнул, и инженеры носят их на рабочей руке как напоминание о своей ответственности перед обществом. Были попытки пересадить эту традицию на американскую почву, но пока без особого успеха.

Стальное кольцо важно, потому что, не обозначая полного профессионального звания, оно реально символизирует принадлежность к инженерии как профессиональному цеху. Сертификация может сыграть похожую роль в ПО, символизируя приверженность к высшим стандартам инженерии ПО.

Если вы подумаете: «Не нужны нам никакие дипломы или стальные кольца», то будете, наверное, правы. Большинство инженеров ПО не станут получать профессиональные лицензии инженеров – свои дипломы – даже после широкого распространения лицензирования. Скорее всего, большинство не станет и сертифицироваться. Но по мере «взросления» инженерии ПО лицензированию и сертификации будет придаваться все более важное значение. Инженер ПО, который хочет продемонстрировать свою приверженность профессии ПО, будет иметь возможность получить лицензию или сертификат, или же и то и другое.

ГЛАВА ДВАДЦАТАЯ

Кодекс профессионала

В каждом из нас есть что-то от Питера Пена, но по мере взросления растет желание сделать что-нибудь на благо общества. Повторю, что реализация этого стремления дает инженеру главное ощущение удовлетворения жизнью.

Сэмюэль С. Флорман

Одним из признаков зрелой профессии является наличие кодекса этики или стандартов профессионального поведения. С юридической точки зрения от профессионалов требуются более высокие стандарты их работы, чем от непрофессионалов, занятых в той же отрасли. Если знакомый сантехник посоветует вам принять таблетку от боли в желудке, а у вас окажется аппендицит, то ничего неэтичного сантехник не совершит. Но если этот же совет вы получите от врача, не осмотревшего вас, то его поведение будет неэтичным.

Кодекс этики устанавливает стандарты поведения в каждой профессии.[1] Дипломированные бухгалтеры должны сдавать трехчасовой экзамен по кодексу этики в своей сфере деятельности. Адвокаты должны сдавать

[1] Например, «Кодекс этики и профессионального поведения» Американского Института Архитекторов на сайте *www.aiaonline.com*; «Кодекс профессионального поведения» Американского Института Дипломированных Бухгалтеров на сайте *www.aicpa.org*; «Кодекс этики инженеров» Национального Общества Профессиональных Инженеров на сайте *www.nspe.org*; «Кодекс этики инженеров» Американского Общества Инженеров-Механиков на сайте *www.asme.org*; а также «Кодекс этики» Института Инженеров Электроники и Электротехники на сайте *www.ieee.org*. Действующие ссылки на все эти кодексы этики имеются на сайте моей компании *www.construx.com/profession*.

экзамен по этике продолжительностью полдня. В сформировавшихся профессиях можно лишиться профессионального статуса или лицензии в случае серьезных нарушений кодекса этики.

Кодекс для кодировщиков

Разработка ПО продолжается уже много лет, обходясь без общепризнанного кодекса этики. В конце 90-х годов XX века совместный комитет организаций ACM и компьютерного общества IEEE начал разработку кодекса этики в инженерии ПО. Кодекс претерпел несколько редакций и рассматривался практикующими разработчиками ПО во всем мире. В 1998 г. Кодекс этики и профессиональной практики был принят как ассоциацией ACM, так и компьютерным обществом IEEE. Он воспроизведен ниже на рис. 20.1. Более детальный вариант кодекса имеется на сайте компьютерного общества *www.computer.org*.

В преамбуле кодекса сформулированы его цели, а далее в 8 пунктах изложены принципы.

Первая основополагающая цель – «инженеры ПО должны служить тому, чтобы анализ, спецификация, проектирование, разработка, тестирование и обслуживание ПО стали выгодной и уважаемой профессией». Другими словами, одна из задач кодекса – способствовать развитию самой про-

Кодекс этики и профессиональной практики инженерии ПО

Инженеры ПО должны служить тому, чтобы анализ, спецификация, проектирование, разработка, тестирование и обслуживание ПО стали выгодной и уважаемой профессией. Верные целям здоровья, безопасности и блага общества, инженеры ПО соблюдают следующие восемь принципов:

1. **Общество**. Инженеры ПО должны действовать в соответствии с интересами общества.
2. **Клиент и работодатель**. Инженеры ПО должны действовать в интересах своего клиента и работодателя согласно интересам общества.
3. **Продукт**. Инженеры ПО должны обеспечить соответствие своих программных продуктов и их последующих модификаций наивысшим профессиональным стандартам.
4. **Решения**. Инженеры ПО должны сохранять непредвзятость и независимость в своих профессиональных суждениях.
5. **Управление**. Менеджеры и руководители инженеров ПО должны придерживаться и содействовать продвижению этического подхода к управлению разработками и обслуживанием ПО.
6. **Профессия**. Инженеры ПО должны поддерживать мораль и репутацию профессии в соответствии с интересами общества.
7. **Коллеги**. Инженеры ПО должны быть справедливыми и поддерживать своих коллег.
8. **Инженеры ПО**. Сами инженеры ПО должны постоянно осваивать свою профессию и способствовать этическому подходу в практической профессиональной деятельности.

Рис. 20.1. *Кодекс этики и профессиональной практики инженерии ПО. Принят ассоциацией ACM и компьютерным обществом IEEE. Является руководством по этике и профессиональному поведению для инженеров ПО*

фессии инженера ПО. Формулировка этой цели неявно предполагает, что инженерия ПО еще не стала «выгодной и уважаемой профессией». По мере ее становления как зрелой профессии формулировка может измениться, и вместо «стали» появится, например, «чтобы анализ... были символом выгодной и уважаемой профессии».

Вторая основополагающая цель состоит в том, что инженеры ПО должны быть «верными целям здоровья, безопасности и блага общества». Это вполне соответствует представлению о том, что инженеры несут скорее ответственность перед обществом в целом, чем перед отдельными его представителями. Кодексы поведения других инженеров аналогичным образом подчеркивают важность защиты благосостояния общества.[1] Две этих цели являются основными, а восемь принципов направлены на их реализацию. Рассмотрим их подробнее.

1. **Общество.** Инженеры ПО должны нести полную ответственность за свою работу и утверждать ПО к выпуску только в том случае, если они твердо уверены в его безопасности и соответствии техническим условиям, если ПО прошло соответствующее тестирование и в конечном итоге будет служить на благо общества. Специалисты ПО должны информировать о любой существующей или потенциальной опасности для отдельных лиц, общества или окружающей среды.
2. **Клиент и работодатель.** Результаты деятельности инженеров ПО непосредственно затрагивают клиентов и работодателей, поэтому разработчики ПО должны защищать их интересы, исключая ситуации, когда эти интересы вступают в конфликт с интересами всего общества. Инженеры обязаны действовать только в рамках своей профессиональной квалификации, хранить конфиденциальную информацию, в своей работе не выходить за пределы области своих знаний и не способствовать интересам, наносящим ущерб их заказчикам или работодателям. Они не должны использовать ПО, полученное незаконным или неэтичным образом. Если инженеры считают, что проект может закончиться неудачей, они должны аргументировать клиенту или работодателю свое мнение.
3. **Продукт.** В своей работе инженеры ПО должны стремиться к высокому качеству, низкой стоимости и разумным срокам исполнения заказа. Они должны показать своим работодателям и заказчикам

[1] Например, «Кодекс этики инженеров» Национального Общества Профессиональных Инженеров на сайте *www.nspe.org*.

возможность компромиссов для достижения баланса между этими тремя показателями. Необходимо дать оценку неопределенности в заложенных показателях качества, смете расходов и сроках исполнения. Инженеры ПО при этом должны следовать соответствующим профессиональным стандартам. Они должны обеспечить надлежащие экспертизу и тестирование ПО перед выпуском в обращение.

4. **Решения.** У настоящих специалистов есть и право, и обязанность, принимая решения, сохранять приверженность высшим профессиональным стандартам, даже когда это противоречит их собственным интересам или интересам их клиентов или работодателей. Утверждать к использованию можно только те продукты, которые адекватно проанализированы и с принципами которых инженеры ПО объективно согласны. Они не будут замешаны в незаконной или бесчестной деятельности: взяточничестве, двойном выставлении счетов или заключении двойных контрактов при наличии конфликта интересов, остающегося нераскрытым полностью.
5. **Управление.** Менеджеры в инженерии ПО должны соблюдать те же профессиональные стандарты, что и другие специалисты ПО, включая положения кодекса этики. Менеджеры должны честно и справедливо относиться к своим сотрудникам. Они должны давать задания инженерам, способным их квалифицированно выполнить, при этом содействуя повышению образования и расширению опыта каждого сотрудника. Они обязаны обеспечить реалистичные количественные оценки расходов, сроков, кадрового обеспечения, качества и других показателей проектов.
6. **Профессия.** Специалисты ПО должны способствовать развитию инженерии ПО как профессии, расширять ее общественное признание. Они должны создавать атмосферу следования кодексу и отказываться от работы в организациях, нарушающих его. Также инженеры должны сообщать о серьезных нарушениях кодекса коллегам, менеджерам или в компетентные организации.
7. **Коллеги.** Инженеры ПО должны помогать своим коллегам следовать кодексу этики. Они должны относиться к ним честно и справедливо, помогать в профессиональном развитии. В ситуациях, выходящих за пределы их компетентности, они должны привлекать других профессионалов, обладающих требуемой квалификацией.
8. **Сами инженеры ПО.** Специалисты ПО должны рассматривать самообразование как приоритет в своей деятельности. Они должны

поддерживать актуальный уровень осведомленности о достижениях в формулировке технических условий, проектировании, написании, обслуживании, тестировании и управлении ПО. Они должны знать действующие стандарты и законодательство, относящиеся к их сфере деятельности.

Преимущества этического кодекса поведения

Этический кодекс обеспечивает широкую поддержку профессиональной инженерии ПО. Он дает работодателям и заказчикам уверенность в профессиональных стандартах и личности инженеров, которые его соблюдают.

Кодекс обеспечивает организациям возможность выразить свою поддержку инженерии ПО. Если компания готова соблюдать кодекс, то обязана обеспечить рабочую среду, в которой этика поведения является приоритетом и инженеры ПО могут следовать ей, не подвергая риску свою карьеру. Это выгодно как самой компании, так и инженерам ПО, которые в ней работают: компания привлекает инженеров с высокими профессиональными стандартами, а они получают возможность реализации в обстановке, где подобные стандарты должным образом оцениваются.

Одним из самых больших преимуществ кодекса является общее направление, которое он задает в смысле этического и профессионального поведения инженеров ПО, что до сих пор отсутствовало начисто. Рассмотрим несколько ситуаций:

- *Проекты, ведущие в тупик*. Не имея этического кодекса, инженеры ПО, считающие, что сроки реализации проекта нереальны, испытывают сомнения, прежде чем сообщать об этом заказчику или своему руководителю. Опираясь на кодекс, в подобной ситуации инженеры обязаны собрать доказательства и документально подтвердить свои опасения. Кодекс гласит, что профессиональным долгом является незамедлительное сообщение руководству или заказчику о своих опасениях.
- *Занижение стоимости разработки с целью получения контракта любой ценой*. В отрасли ПО распространена практика коммерческих предложений заказчикам с нереалистично низкими расценками. Разработчикам ПО, возможно, не слишком нравится такое поведение, но многие не готовы пойти наперекор своим боссам и отказаться занизить стоимость разработки в заявке. Согласно кодексу

инженеры ПО должны обеспечить реалистичность смет расходов и одобрять документы, только если они с ними согласны. Кодекс также призывает инженеров ПО сделать свою профессию достойной уважения, например не участвовать в сделках с заниженной стоимостью. Соблюдающий этические принципы инженер ПО должен отказаться утверждать такие коммерческие предложения.

- *Разработка ПО по принципу «напишем и исправим».* Недостаточно информированные заказчики и руководители часто настаивают на применении разработчиками подхода «напишем и исправим». Последние сознают неэффективность этого подхода, но после споров с заказчиками и руководством многие капитулируют: «Пусть эта контора пожнет то, что заслужила». Однако следование данному принципу противоречит этическому долгу инженера ПО создавать высококачественные продукты по приемлемым ценам и в разумные сроки. Продолжающееся использование названного метода также компрометирует инженерию ПО как профессию, поэтому следующие этическим принципам разработчики ПО должны отказаться от него.
- *Застой знаний.* Для того чтобы оставаться в курсе всего нового в разработке ПО, требуется много времени, и многие разработчики даже не пытаются делать это. Один из издателей утверждал, что средний разработчик ПО читает в год меньше одной книги по своей профессии и не выписывает ни одного профессионального журнала [37][1]. Возможно, проблема не связана с этикой, но уж наверняка связана с *профессиональным поведением*. Невозможно работать на уровне профессионала, не интересуясь последними достижениями в своей отрасли. Не занимаясь непрерывным самообразованием, можно продолжать работать в отрасли ПО на каком-то уровне, однако согласно кодексу этики и профессионального поведения нельзя при этом быть специалистом ПО.

Вне действия кодекса инженерам ПО приходится полагаться на собственные суждения и оценки, сталкиваясь с этическими дилеммами. Инженеры, придерживающиеся кодекса, будут знать, что им не придется в одиночку отстаивать свои позиции.

---

[1] Том Демарко и Тимоти Листер «Человеческий фактор: успешные проекты и команды», 2-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2005.

В некоторых случаях ситуация не столь однозначна, как в вышеприведенных примерах. Интересы клиента-заказчика будут противоречить интересам общественного блага. Или интересы работодателя могут противоречить интересам коллег по цеху разработчиков ПО. Кодекс не может предусмотреть все этические дилеммы, но призывает инженеров ПО принимать решения, исходя из высших нравственных критериев, соблюдая при этом дух кодекса.

Достижение совершеннолетия

Кодекс информирует общество, включая клиентов и руководящее звено, о том, чего следует ожидать от специалистов ПО. Разумеется, его существование имеет смысл, только если работодатели и заказчики смогут рассчитывать на соблюдение кодекса профессионалами инженерии ПО. В каждой профессии нужно иметь дисциплинарные средства, чтобы призвать к порядку работников, не соблюдающих профессиональные стандарты. Не имея подобных средств, можно подорвать доверие к профессии в результате постепенного ее разрушения работниками, не придерживающимися профессиональных принципов. Сегодня ни компьютерное общество IEEE, ни ACM, ни любая другая организация не пользуются полным авторитетом, чтобы принудить своих работников следовать кодексу. Соблюдение его добровольно. В долгосрочном плане, однако, инженерия ПО последует по тому же пути, что и другие профессии: полноценный профессиональный статус и соблюдение кодекса станут ее неотъемлемыми частями. Кодекс будет обязательным для исполнения, и это будет выгодно и специалистам ПО, и их работодателям и заказчикам, и обществу в целом.

Сэмюэль Флорман сказал, что «по мере взросления у нас растет желание сделать что-то на благо общества». Он говорил о личностях, но его слова можно отнести и к нашей теме. Кодекс этики и профессионального поведения, который обращает особое внимание на ответственность по отношению к профессии и вклад в общественное благо в целом, является одним из главных признаков начала взросления инженерии ПО.

ГЛАВА ДВАДЦАТЬ ПЕРВАЯ

Алхимия

Вопрос: Назовите, пожалуйста, самые перспективные идеи и методики инженерии ПО, появляющиеся на горизонте?

Ответ: Думаю, что они не на горизонте. Они здесь, с нами, уже много лет. Просто никто не извлекает из них всей пользы.

Дэвид Л. Парнас [2]

Как превратить железо в золото? Последователи царя Мидаса во всех областях не устают задавать этот вопрос. Инженерия ПО уникальна в своем роде, поскольку может реализовать эту мечту алхимиков, ускорив внедрение давно уже понятых и прекрасно зарекомендовавших себя методик, которые все еще не получили должного распространения. Мы можем превратить второсортный свинец в золото высшей пробы.

Зачем передавать технологии практикам

Отрасль инженерии ПО располагает надежными методиками планирования и управления проектами, выработки технических требований, проектирования, создания, обеспечения качества и совершенствования процесса. Проблема лишь в том, что лишь немногие практики о них знают, а применяют их и еще меньше. В табл. 21.1 приведены некоторые примеры успешных методик, в применении которых ведущие организации ПО накопили значительный опыт, но которые, насколько я могу судить по собственному опыту консультирования и по различным отраслевым обзорам, используются лишь очень тонкой прослойкой организаций-разработчиков ПО.

Аналитики обнаружили, что обычно инновации – новые методики – проходят путь от освоения технологии до применения за 10–15 лет [113], [124]. Если это так, то процесс освоения новых технологий в отрасли разработки ПО существенно нарушен. Большинство методик, перечисленных в табл. 21.1, были изложены на бумаге 15 лет назад и даже больше. Почему же их не используют?

Таблица 21.1. Результативные, но редко применяемые методики ПО [83]

Методика	Год первой публикации или коммерческого предложения
Планирование и управление проектом	
• Средства автоматического оценивания	1973 [64]
• Эволюционная наработка и сдача продукта	1988 [51]
• Замеры	1977 [50]
• Производительная среда	1984 [12], [36]
• Планирование управления рисками	1981[a]
Разработка технических требований	
• Совет по изменениям	1978[b]
• Опережающее прототипирование пользовательских интерфейсов	1975 [19]
• Привлечение заказчика к участию в разработке через совместные рабочие семинары (сеансы совместной разработки приложений)	1985[c]
Методики проектирования	
• Сокрытие информации	1972 [104]
• Встраивание процесса изменений	1979 [105]

[a] Управление рисками значительно старше, чем сама отрасль ПО, но работы по управлению рисками, связанные конкретно с ПО, стали появляться после выхода труда Ф. У. Макфарлана (F. W. McFarlan) [88].

[b] Советы по управлению изменениями значительно старше, чем отрасль ПО, однако книги, связанные конкретно с ПО, и работы о советах по изменениям в ПО (или по управлению конфигурацией ПО) стали появляться после работы Эдварда Х. Берсоффа (Edward H. Bersoff) [9].

[c] Семинары по совместной разработке ПО с привлечением заказчика (JAD) практиковались в IBM еще в 1977 г., но впервые сообщения о них появились в печати в работе Гэри Раша (Rush, Gary) [120].

Таблица 21.1 (продолжение)

Методика	Год первой публикации или коммерческого предложения
Методики написания ПО	
• Контроль исходного кода	1980[a]
• Пошаговая интеграция	1979 [93]
Методики обеспечения качества	
• Тестирование с проходом всех ветвей участков программы	1979 [93]
• Экспертизы	1976 [42]
Совершенствование процесса разработки	
• Модель зрелости разработки ПО Института инженерии ПО	1987 [58]
• Группы по процессам разработки ПО	1989 [59]

[a] Различный инструментарий контроля исходной программы существовал и до 1980 г., но одно из первых упоминаний в печати появилось в работе [10].

Распространение инноваций

Ответ на этот вопрос одновременно и легок, и труден. Процесс проникновения инноваций в повседневную практику изучался довольно широко. Основополагающий труд в этой области принадлежит Эверрету М. Роджерсу и называется «Diffusion of Innovations» (Распространение инноваций). Впервые работа увидела свет в 1962 г. [119], а в четвертом издании, опубликованном в 1995 г., автор отметил, что свыше 3,5 тысяч статей и книг, посвященных проблеме распространения инноваций, было опубликовано между первым и четвертым изданиями книги.

Э. М. Роджерс описал процесс принятия инноваций, исходя из типизации людей, воспринимающих их. Он разделил их на пять категорий: инноваторы-энтузиасты; распространители технологий; группы принятия инноваций; эксперты; раннее большинство; позднее большинство и медлящие. На рис. 21.1 показаны как относительные размеры каждой из категорий, так и последовательность восприятия ими инноваций.

Инноваторы – это энтузиасты со склонностью к экспериментам. Они любят испытывать новые технологии независимо от степени их готовности и сопутствующего им риска. Их не волнует значительная степень неоп-

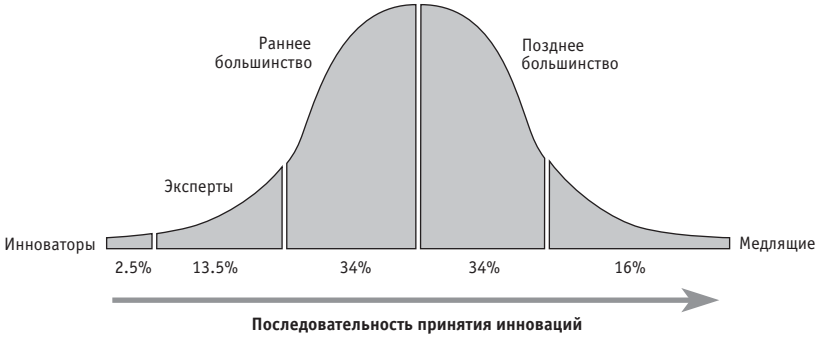

Рис. 21.1. *Инновации распространяются постепенно. У различных групп, принимающих инновацию, различные потребности и разные критерии ее оценки. Источник: работа [119]*

ределенности при использовании еще не сложившихся новых технологий и методик. Поскольку они рискуют, их часто преследуют неудачи. Поэтому люди, принадлежащие к другим категориям, могут не уважать инноваторов.

Эксперты – это лидеры в своей организации, их мнение уважают. Они опережают остальных в принятии инноваций, но ровно настолько, чтобы служить примером и увлекать других.

Люди из категории *раннего большинства* более рассудительны и внимательны в принятии инноваций, чем эксперты, и относятся к одной из двух крупнейших групп. Чтобы решиться на использование инноваций, им нужно больше времени, чем экспертам. Они обычно следуют по стопам категории экспертов.

Позднее большинство настроено по отношению к инновациям скептически. Оно ведет себя очень осторожно и принимает инновации только после того, как они многими опробованы. Эти люди не убеждены, что инновации приводят к улучшениям или что они применимы в их случае. Принятие ими инноваций может зависеть от желания не отставать от коллег.

Медлящие – последние в принятии инноваций; они скорее смотрят в прошлое, чем в будущее. Люди этой категории чрезвычайно осторожны, принимая новое.

Пропасть

Работы Э. М. Роджерса были продолжены в середине девяностых годов XX века Джеффри Муром (Geoffrey Moore) в книге [92]. Мур заметил, что

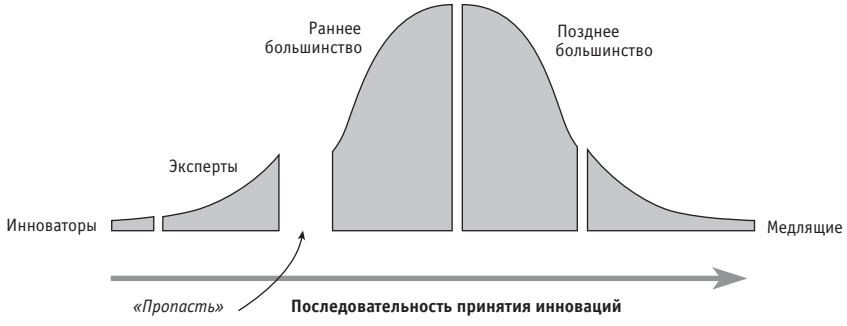

Рис. 21.2. *Одна из трудностей распространения технологии – необходимость преодолеть пропасть, разделяющую запросы экспертов и раннего большинства. Источник: работа [92]*

различия в стилях принятия решений создают водоразделы между категориями. Аргументы в пользу принятия инновации, которые весьма убедительны для инноваторов-энтузиастов, не обязательно убеждают экспертов.

Но самое важное наблюдение, сделанное Муром, заключается в неоднородности водоразделов. По его мнению (рис. 21.2), разрыв между экспертами и ранним большинством значительно шире, чем между остальными, и его с полным основанием можно назвать пропастью.

Несколько жестких вопросов

Одна из причин медленного внедрения инноваций состоит в том, что не все они эффективны на практике! Не все инновации полезны, поэтому у раннего большинства, позднего большинства и медлящих достаточно оснований для осторожности. Думая о внедрении инновации, они задаются серьезными вопросами, например [126]:

- Дают ли результаты экспериментов основания для окончательного вывода, что инновация будет работать на практике?
- Являются ли достигнутые успехи результатом инновации, или они связаны с практической работой конкретных людей?
- Является ли инновация полной или требует доработки/расширения перед внедрением?
- Сопутствуют ли инновации дополнительные издержки (расходы на обучение, документацию), нивелирующие выигрыш от ее внедрения в долгосрочном плане?

- Если инновация создавалась в исследовательской среде, применима ли она в обычных практических условиях?
- Замедляет ли инновация в целом работу программистов?
- Возможно ли некорректное применение инновации?
- Имеются ли данные о рисках, связанных с использованием данной инновации?
- Известно ли, как интегрировать инновацию в существующие методики?
- Надо ли внедрять инновацию полностью, чтобы получить от этого ощутимую выгоду?

Лишь по очень немногим методикам инженерии ПО собраны и достаточно широко распространены все данные, необходимые для того, чтобы практики ПО могли дать ответы на эти вопросы. В результате практические методики разработки ПО, указанные в табл. 21.1, застряли по левую сторону от «пропасти». Эксперты применяют эти методики уже больше 15 лет, а группы раннего большинства в основном остаются в неведении. Количественные данные по категориям принятия инноваций близки к данным о количестве проектов ПО, в которых процветает методика «напишем и исправим», – примерно в 75% проектов по-прежнему применяется этот подход или его ближайшие собратья, и примерно такой же процент людей относится к категориям справа от «пропасти».

В чем причина такой ситуации? В парадигме Э. М. Роджерса одна из причин того, что инновации быстрее доходят до инноваторов и экспертов, заключается в том, что эксперты обычно располагают большими ресурсами – они чаще других могут позволить себе дорогостоящие ошибки. Остальные категории более осторожны, в частности из-за ограниченности ресурсов. Однако, как говорилось в главе 12, в данном случае ограничен не денежный ресурс, а *время*. Типичным методикам типа «напишем и исправим» сопутствуют существенные нарушения сроков, а связанные с этим сверхурочные не оставляют никакого времени для изучения и взятия на вооружение более эффективных инновационных методик.

В чем риск?

Здравый смысл подсказывает, что если вы готовы рисковать, то находитесь слева от пропасти, а если нет – то справа. Однако такие умозаключения плохо укладываются в существующую картину инженерии ПО.

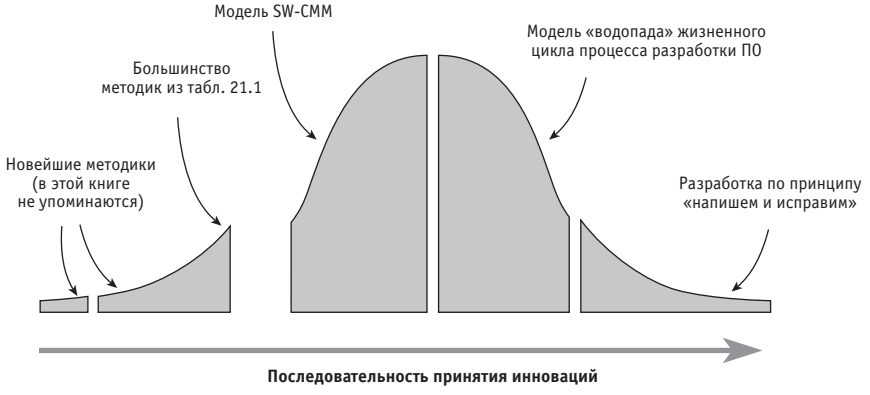

Рис. 21.3. *Различные методики разработки ПО находятся на разных стадиях принятия новых технологий. Методики, названные в табл. 21.1, готовы к преодолению пропасти. Методика СММ в разработке ПО получила широкое распространение в одних отраслях, но почти неизвестна в других (т. е. в каких-то отраслях водораздел пройден, а в каких-то нет). Устаревшие методики (аналогичные принципу «напишем и исправим») постепенно исчезают (или, по крайней мере, должны)*

Как показано на рис. 21.3, методики из табл. 21.1 сегодня применяются инноваторами и экспертами. На этом рисунке также указаны некоторые конкретные методики, играющие роль точек отсчета. Модель СММ Института инженерии ПО, по-видимому, как раз преодолевает пропасть. Хорошо известная модель «водопада» жизненного цикла процесса разработки попадает в зону позднего большинства, а разработка по принципу «напишем и исправим» прочно увязла в зоне медлящих.

Следует признать, что после конференции НАТО по инженерии ПО в 1968 г. передовые технологии сделали огромный скачок. Но существует все еще очень много организаций, применяющих методики десяти-двадцатилетней давности и даже более отсталые, которые вполне могут быть заменены современными. Отрасль столкнулась с проблемой медленного принятия инноваций. Уже говорилось, что организации, использующие устарелые методики, сильно рискуют превысить бюджет, затянуть сроки и просто оказаться у разбитого корыта в связи с отменой проектов. Этот факт отмечен здесь повторно, чтобы подчеркнуть, что организации, занимающие в области ПО позицию в одной из категорий справа от пропасти, *в данный момент не минимизируют риски*. На бесконечные ремонты старого автомобиля можно потратить больше, чем на покупку нового. Точно

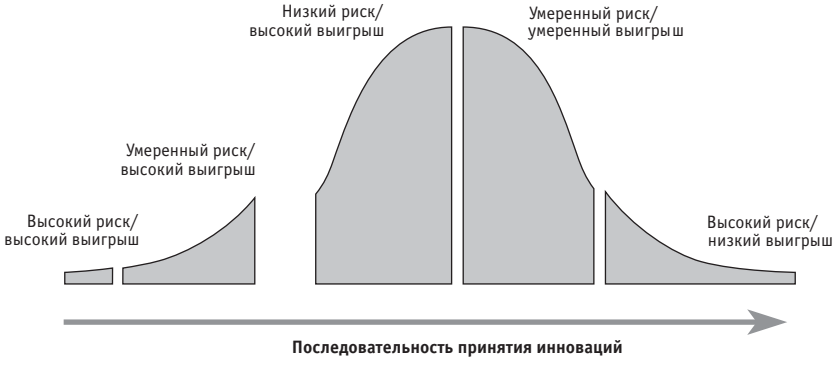

Рис. 21.4. *С учетом скорости совершенствования методик разработки ПО традиционное соотношение риска и выигрышей сместилось. Риски высоки на обоих краях кривой принятия инноваций*

так же и риски при использовании устаревших методик разработки ПО могут быть выше, чем при переходе к новым более совершенным методикам.

В какой точке цикла принятия технологии лучше всего находиться? Как видно из рис. 21.4, на левом краю риск принятия перспективной, но еще не зарекомендовавшей себя методики оправдан, поскольку обещает громадный выигрыш в случае успеха. На правом краю риск от применения устаревших методик столь же высок, но не сулит никакого выигрыша и поэтому не оправдан.

Ситуация с распространением технологий в отрасли разработки ПО необычна. Многие новейшие методики, подтвердившие свою ценность, такие как указанные в табл. 21.1, «скопились» на левом краю пропасти, разделяющей экспертов и категории справа от пропасти, и готовы преодолеть ее в организациях, решивших отказаться от пристрастия к разработке по принципу «напишем и исправим» и ему подобным ложно выигрышным методикам. Данная ситуация необычна тем, что в привычных обстоятельствах риски для экспертов были бы выше, например, как если бы передовые врачи, испытав пенициллин, доказали его эффективность и внедрили в свою практику, а 75% врачей по-прежнему прописывали бы пиявки и горчичники. Если эти средства применяются в начале двадцать первого века, то риск непринятия инноваций выше, чем при их использовании.

Если ваша организация входит в группу раннего или позднего большинства или медлящих, то риск можно снизить, приняв некоторые из современных методик, указанных в табл. 21.1. Подход по принципу «напи-

шем и исправим» рискован. Использование модели «водопада» в разработке ПО тоже рискованно. Принятие модели СММ не лишено риска, но он не настолько высок, как у этих двух устаревших подходов. Отраслевой опыт, на который я постоянно здесь ссылаюсь, подтверждает справедливость этого утверждения. Трудность состоит в том, чтобы широко распространить этот вывод.

Опыт работы представителей на местах по программе расширения консультационной деятельности в сельском хозяйстве США

В сфере распространения инноваций одной из самых успешных программ в мире считается программа службы расширения консультирования по сельскому хозяйству в США. Как говорит автор одной из работ по этой теме, «невозможно сказать и десятка слов по распространению инноваций, не упомянув консультационный сервис в сельском хозяйстве».[1] Этот сервис включает три части:

- *Научно-исследовательская система* – научные работники-исследователи на экспериментальных участках во всех 50 штатах США разрабатывают инновации, которые затем распространяются повсеместно.
- *Консультанты в штатах* – увязывают исследовательские работы с представителями на местах.
- *Консультационные представители на местах* – работают с фермерами и другими производителями сельскохозяйственной продукции на местах, помогая им выбирать инновации, подходящие для их нужд. Именно они отвечают на вопросы практиков, например: «Является ли инновация полной или требует доработки/расширения перед внедрением?» или «Являются ли достигнутые успехи результатом внедрения инновации, или же они связаны с практикой работы людей?»

Эта программа в сельском хозяйстве придает особо важное значение кооперации трех составных частей. Ученые-исследователи получают вознаграждение за опубликование результатов исследований в нужном для фермеров направлении. Работа консультантов в штатах оценивается по

[1] Ивланд Дж. Д. (Eveland J. D.), цит. по: [119]. Информация о распространении инноваций в сельском хозяйстве почерпнута из этой же книги.

тому, насколько хорошо они увязывают свои исследования с проблемами фермеров.

Ежегодные инвестиции в программу по распространению сельхозинноваций примерно равны ежегодным инвестициям в исследования. Ни одна федеральная целевая программа, кроме этой, не тратит больше нескольких процентов своего бюджета на распространение результатов, и ни одна не достигла таких успехов в изменении сложившейся практики.

Опыт отрасли ПО также дает несколько аргументов в пользу распространения инноваций. Одна из лучших организаций ПО – Лаборатория инженерии ПО NASA – установила, что представление результатов ее программы измерений показателей и анализа в формате руководств и курсов обучения стало ключевым фактором успеха завоевавшей множество наград программы совершенствования процесса разработки ПО [139].

Распространение инноваций в отрасли ПО необходимо, и в ней уже существует зародыш такой программы. При федеральном финансировании был создан Институт инженерии ПО (SEI), призванный «играть ведущую роль в развитии практического уровня инженерии ПО с целью повысить качество зависящих от ПО систем».[1] Фактически Институт SEI выполняет ту же роль, что научно-исследовательская система в программе распространения инноваций сельского хозяйства. Однако, имея лишь 300 сотрудников на 2,9 миллиона занятых в отрасли ПО, программа этого института находится на начальных стадиях по сравнению с программой в сельском хозяйстве, в которой около 17 тысяч сотрудников обслуживают около 3,8 миллионов занятых в АПК [129].

Э. Роджерс указывал, что многие правительственные агентства пытались скопировать модель программы расширения консультаций в сельском хозяйстве, но потерпели неудачу, поскольку, кроме всего прочего, они не сформировали институт представителей на местах, как это было сделано по программе в сельском хозяйстве для обеспечения внедрения на местном уровне. В анализе Роджерса много внимания уделено объяснению причин пока еще ограниченного влияния Института SEI на коммерческую практику. Институт был создан Министерством обороны США; и документация, и материалы, создаваемые в институте, имеют выраженный военный оттенок. Неудивительно, что раньше всех выгоды распространения институтом технологий получили военные подрядчики и правительственные агентства [54].

---

[1] См. сайт Института инженерии ПО по адресу *www.sei.cmu.edu*.

В отрасли ПО существует много подотраслей со своими нуждами и своим сленгом. Среди них бизнес-системы, веб-разработки, игры, ПО для медицинской аппаратуры, системное ПО, производство компьютеров, встроенные системы, аэрокосмические приложения и многие другие. Если сложить неспособность выразить инновации инженерии ПО в терминах, понятных для каждой конкретной подотрасли, с общей нехваткой глубокого образования в этой сфере, то медленное продвижение инноваций станет понятным.

Практики не примут инновации, пока не получат ответы на интересующие их вопросы в понятных для них терминах. Чтобы добиться эффективности внедрения технологий в практику, надо чтобы либо государство, либо частный сектор финансировали выполнение функций, близких к функциям консультантов в штатах и представителей на местах в сельхозпрограмме. Требования проектов ПО не будут так уж сильно меняться в зависимости от местности, как в случае сельского хозяйства. Но они будут меняться в зависимости от подотрасли, так что инженерия ПО вполне могла бы выиграть от сотрудников по связям на местах, которые увязывали бы общие исследования в области инженерии ПО с запросами конкретных подотраслей.

Другие меры, которые могут способствовать ускорению внедрения технологий в практику, включают элементы формирования профессии, которые являются темами данной книги: программы подготовки специалистов без степени по инженерии ПО, дипломы профессионалов, требования непрерывного профессионального образования, кодекс профессионального поведения.

Принижающая роль прогресса

Несколько лет назад мне случилось заехать в небольшой сельский городок не ради общения с местным представителем программы расширения сельского хозяйства, а чтобы увидеться с коллегой по инженерии ПО, с которым раньше лично не встречался. Сразу после нашего знакомства он спросил меня: «Наверное, нужно быть очень храбрым человеком, чтобы в вашем возрасте написать столь объемную книгу, как «Code Complete»?[1]

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

Как бы мне ни нравилось, что меня назвали храбрым, я не могу согласиться с таким высказыванием. Моя книга «Code Complete» является примером передачи научных и инженерных знаний от поколения к поколению и того, как в конечном итоге идет развитие знаний. Пионеры в области инженерии ПО – Виктор Бэсили, Барри Боем, Ларри Константин, Билл Кертис, Том Демарко, Том Гилб, Кейперс Джоунз, Харлан Миллз, Дэвид Парнас и другие – прилагали массу усилий для выработки передовых концепций из крупиц несформированных знаний. Они трудились в окружении ошибочных теорий, противоречивых данных и разрозненных или просто несуществующих разработок прошлых лет. Их последователи воспользовались работами пионеров и смогли избежать заблуждений и неверных предположений. «Новые люди», которым не пришлось блуждать в поисках краеугольных концепций, иногда могут проще и доступнее разъяснить новаторские идеи основоположников, чем те могли бы сделать это сами. Ларри Константин проделал работу, из которой родилось структурное проектирование. Эд Йордон проанализировал работы Л. Константина по структурному проектированию, но результаты анализа так и не были поняты большинством читателей [144]. И только после того, как Мейир Пейдж-Джоунс [103] прокомментировал пояснения Э. Йордона к работе Л. Константина, эти идеи наконец-то стали доступны широкому кругу рядовых практиков [27]. Затем идеи структурного проектирования были интегрированы в объектно-ориентированное проектирование. И цикл возобновился.

Проходит время, и знания, которыми первопроходцы овладевали в течение всей своей деятельности, студенты осваивают за пару семестров. Книгу «Code Complete» я писал три с половиной года. Когда-нибудь разработчик ПО, возможно еще «храбрее» меня, напишет книгу значительно лучше моей всего за несколько месяцев. Именно так свинец медленно превращается в золото в отраслях интенсивного использования знаний, таких как инженерия ПО. Так и должно быть.

Библиография

[1] Abran, Alain, et al., «Guide to the Software Engineering Body of Knowledge: Trial Version (Version 1.00)», IEEE Computer Society, 2001.

[2] «ACM Fellow Profile: David Lorge Parnas», *ACM Software Engineering Notes*, May 1999, pp. 10–14.

[3] Anthes, Gary H., «IRS Project Failures Cost Taxpayers $50B Annually», *Computerworld*, October 14, 1996.

[4] Bach, James, «Enough about Process: What We Need Are Heroes», *IEEE Software*, March 1995, pp. 96–98.

[5] Baines, Robin, «Across Disciplines: Risk, Design, Method, Process, and Tools», *IEEE Software*, July/August 1998, pp. 61–64.

[6] Baker, F. Terry, «Chief Programmer Team Management of Production Programming», *IBM Systems Journal*, vol. 11, no. 1, 1972, pp. 56–73.

[7] Baker, F. Terry, and Harlan D. Mills, «Chief Programmer Teams», *Datamation*, Volume 19, Number 12 (December 1973), pp. 58–61.

[8] Beck, Kent, «Extreme Programming Explained: Embrace Change», Reading, MA: Addison-Wesley, 1999.

[9] Bersoff, Edward H., «Proceedings of the Software Quality and Assurance Workshop», a joint Publication of ACM *Performance Evaluation Review*, vol. 7, nos. 3 & 4, and ACM *Software Engineering Notes*, vol. 3, no. 5 (1978).

[10] Bersoff, Edward H., et al., «Software Configuration Management», Englewood Cliffs, NJ: Prentice Hall, 1980.

[11] Boehm, Barry W., «Software Engineering Economics», Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.

[12] Boehm, Barry W., et al., «A Software Development Environment for Improving Productivity», *IEEE Computer*, June 1984, pp. 30–44.

[13] Boehm, Barry W., «Improving Software Productivity», *IEEE Computer*, September 1987, pp. 43–57.

[14] Boehm, Barry, et al., «Software Cost Estimation with Cocomo II», Boston, MA: Addison-Wesley, 2000.

[15] Böhm, C., and G. Jacopini «Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules», *Communications of the ACM*, May 1966, pp. 366–371.

[16] Bostrom, R. P., and K. M. Kaiser, «Personality Differences within Systems Project Teams», *Proceedings of the 18th Annual Computer Personnel Research Conference*, ACM No. 443810, 1981.

[17] Britcher, Robert N., «Why (Some) Large Computer Projects Fail» in Glass, Robert L., «Software Runaways», Englewood Cliffs, NJ: Prentice Hall, 1998.

[18] Bronson, Po, «Manager's Journal», *Wall Street Journal*, February 9, 1998.

[19] Brooks, Frederick P., Jr., «The Mythical Man-Month, Reading», MA: Addison-Wesley, 1975.

[20] Brooks, Frederick P., Jr., «No Silver Bullets – Essence and Accidents of Software Engineering», *Computer*, April 1987, pp. 10–19.

[21] Brooks, Frederick P., Jr., «The Mythical Man-Month», Anniversary Edition, Reading, MA: Addison-Wesley, 1995.[1] Первое издание вышло в 1975.

[22] Bylinsky, Gene, «Help Wanted: 50,000 Programmers», *Fortune*, March 1967, pp. 141ff.

[23] Carnegie Mellon University/Software Engineering Institute, «The Capability Maturity Model: Guidelines for Improving the Software Process», Reading, MA: Addison-Wesley, 1995.

[24] Cole, Andy, «Runaway Projects – Cause and Effects», *Software World*, Vol. 26, no. 3, pp. 3–5.

[25] Constantine Larry, Myers Glenford, and Stevens Wayne, «Structured Design», *IBM Systems Journal*, No. 2, 1974, pp. 115–139.

[26] Constantine, Larry, «Under Pressure», *Software Development*, October 1995, pp. 111–112.

[27] Constantine, Larry, «Constantine on Peopleware», Englewood Cliffs, NJ: Yourdon Press, 1995.

[28] Constantine, Larry, «Re: Architecture», *Software Development*, January 1996, pp. 87–88.

[1] Фредерик Брукс «Мифический человеко-месяц или как создаются программные системы». – Пер. с англ. – СПб.: Символ-Плюс, 2000.

[29] «Continuing Education Workshop», *The Washington Board Journal*, Board, Winter/Spring 1999, p. 10.

[30] Conway, M. E., «How Do Committees Invent?» *Datamation*, vol. 14, no. 4, 1968, pp. 28–31.

[31] «Criteria for Accrediting Computing Programs: Effective for Evaluations During the 2003-2004 Accreditation Cycle», Computing Accreditation Commission (CAC) of the Accreditation Board for Engineering and Technology, Inc. (ABET), Baltimore, MD, November 2, 2002.

[32] «Criteria for Accrediting Engineering Programs», Accreditation Board for Engineering and Technology, Inc., November 1, 1998.

[33] Curtis, Bill, «Substantiating Programmer Variability», *Proceedings of the IEEE*, vol. 69, no. 7, 1981.

[34] Davis, Alan M., «201 Principles of Software Development», New York: McGraw-Hill, 1995.

[35] DeMarco, Tom, «Structured Analysis and System Specification», NJ: Prentice Hall, 1979.

[36] DeMarco, Tom, and Timothy Lister, «Programmer Performance and the Effects of the Workplace», in *Proceedings of the 8th International Conference on Software Engineering*, August 1985.

[37] DeMarco, Tom, and Timothy Lister, «Peopleware: Productive Projects and Teams», 2d Ed., New York: Dorset House, 1999.[1]

[38] DeMarco, Tom, «Certification or Decertification», *Communications of the ACM*, July 1999, p. 11.

[39] Diaz, Michale, and Jeff King, «How CMM Impacts Quality, Productivity, Rework, and the Bottom Line», *CrossTalk*, vol. 15, no. 3 (March 2002), pp. 9–14.

[40] Dijkstra, Edsger, «GoTo Statement Considered Harmful», *Communications of the ACM*, Vol. 11, 1968, pp. 148ff. См. также по адресу: *www.cs.utexas.edu/users/EWD/ewd02xx/EWD215.PDF*.

[41] Duncan, W. R., «A Guide to the Project Management Body of Knowledge», Upper Darby, PA: Project Management Institute, 1996.

[42] Fagan, M. E., «Design and Code Inspections to Reduce Errors in Program Development», *IBM Systems Journal*, v. 15, no. 3, 1976, pp. 182–211.

[43] Fetzer, Daniel T., «Making Investment Decisions for Software Process Improvement», *DACS Software Tech News*, November 2002, pp. 19–22.

[1] Том Демарко и Тимоти Листер «Человеческий фактор: успешные проекты и команды», 2-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2005.

[44] Fishman, Charles, «They Write the Right Stuff», *Fast Company*, December 1996.

[45] Florman, Samuel C., «The Existential Pleasures of Engineering», 2d Ed., NY: St. Martin's Griffin, 1994.

[46] Ford, Gary and Gibbs, Norman E., «A Mature Profession of Software Engineering», SEI, CMU, CMU/SEI-96-TR-004, January 1996.

[47] Frosch, Robert A., «A New Look at Systems Engineering», *IEEE Spectrum*, September 1969.

[48] Gibbs, W. Wayt, «Software's Chronic Crisis», *Scientific American*, September 1994, pp. 86–95.

[49] Gibbs, W. Wayt, «Command and Control: Inside a Hollowed-Out Mountain, Software Fiascoes – and a Signal Success», *Scientific American*, August 1997, pp. 33–34.

[50] Gilb, Tom, «Software Metrics», Cambridge, MA: Winthrop Publishers, 1977.

[51] Gilb, Tom, «Principles of Software Engineering Management», Wokingham, England: Addison-Wesley, 1988.

[52] Glass, Robert L., «Software Creativity», Englewood Cliffs, NJ: Prentice Hall PTR, 1994.

[53] Glass, Robert L., «Software Runaways», Englewood Cliffs, NJ: Prentice Hall, 1998.

[54] Hayes, Will, and Dave Zubrow, «Moving On Up: Data and Experience Doing CMM-Based Process Improvement», CM/SEI-95-TR-008, August 1995.

[55] Hecker, Daniel E., «Occupational employment projections to 2010», *Monthly Labor Review*, November 2001, vol. 124, no. 11.

[56] Herbsleb, James, et al., «Benefits of CMM Based Software Process Improvement: Initial Results», Pittsburgh: Software Engineering Institute, Document CMU/SEI-94-TR-13, August 1994.

[57] Herbsleb, James, et al., «Software Quality and the Capability Maturity Model», *Communications of the ACM*, June 1997, pp. 30–40.

[58] Humphrey, Watts S., and W. L. Sweet, «A Method for Assessing the Software Engineering Capability of Contractors», Report CMU/SEI-87-TR-23, Pittsburgh: Software Engineering Institute, 1987.

[59] Humphrey, Watts S., «Managing the Software Process», Reading, MA: Addison-Wesley, 1989.

[60] Humphrey, Watts S., «Winning with Software: An Executive Strategy», Boston, MA: Addison-Wesley, 2001.

[61] Hutchings, Edward, «Surely You're Joking, Mr. Feynman!», New York: W. W. Norton & Company, Reprint Edition, 1997.

[62] Johnson, Jim, «Turning Chaos into Success», *Software Magazine*, December 1999, pp. 30–39.

[63] Jones, Capers, «Programming Productivity», New York: McGraw-Hill, 1986.

[64] Jones, Capers, «Assessment and Control of Software Risks», Englewood Cliffs, NJ: Yourdon Press, 1994.

[65] Jones, Capers, «Gaps in Programming Education», *IEEE Computer*, April 1995, pp. 70–71.

[66] Jones, Capers, «Patterns of Software Systems Failure and Success», Boston, MA: International Thomson Computer Press, 1996.

[67] Jones, Capers, «Applied Software Measurement: Assuring Productivity and Quality», 2d Ed., New York: McGraw-Hill, 1997.

[68] Jones, Capers, «Software Assessments, Benchmarks, and Best Practices», Boston, MA: Addison-Wesley, 2000.

[69] Kaner, Cem, «Computer Malpractice», *Software QA*, Volume 3, no. 4, 1997, p. 23.

[70] Kennedy, Ken, and Moshe Y. Vardi, «A Rice University Perspective on Software Engineering Licensing», *Communications of the ACM*, November 2002, pp. 94–95.

[71] Knight, John C., and Nancy Leveson, «Should Software Engineers Be Licensed?» *Communications of the ACM*, November 2002, pp. 87–90.

[72] Knuth, Donald, «The Art of Computer Programming, Volume 3: Sorting and Searching», Reading, MA: Addison-Wesley, 1973, p. 419.

[73] Krantz, Les, «Jobs Rated Almanac», NY: St. Martin's Press, 1999.

[74] Krasner, Herb, «Accumulating the Body of Evidence for the Pay off of Software Process Improvement – 1997», November 19, 1997 (unpublished paper).

[75] Kruchten, Philippe, «The Rational Unified Process: An Introduction», 2d Ed., Boston, MA: Addison-Wesley, 2000.[1]

[76] Kuhn, Thomas S., «The Structure of Scientific Revolutions», 3d Ed., Chicago: The University of Chicago Press, 1996.

[77] Lakhanpal, B., «Understanding the Factors Influencing the Performance of Software Development Groups: An Exploratory Group-Level Analysis», *Information and Software Technology*, 35 (8), 1993, pp. 468–473.

[78] Lawlis, Dr. Patricia K., Capt. Robert M. Flowe, and Capt. James B. Thordahl, «A Correlational Study of the CMM and Software Development Performance», *Crosstalk*, September 1995.

[79] Lederer, Albert L., and Jayesh Prasad, «Nine Management Guidelines for Better Cost Estimating», *Communications of the ACM*, February 1992, pp. 51–59.

[1] Ф. Кратчен «Введение в Rational Unified Process», 2-е издание, Вильямс, 2002.

[80] Lowell, Bill, and Angela Burgess, «A Moving Target: Studies Try to Define the IT Workforce», *IT Professional*, May/June 1999.

[81] Lyons, Michael L., «The DP Psyche», *Datamation*, August 15, 1985, pp. 103–109.

[82] McCalla, Gord, «Software Engineering Requires Individual Professionalism», *Communications of the ACM*, November 2002, pp. 98–101.

[83] McConnell, Steve, «Rapid Development», Redmond, WA: Microsoft Press, 1996.

[84] McConnell, Steve, «Software Project Survival Guide», Redmond, WA: Microsoft Press, 1997.[1]

[85] McConnell, Steve, «How to Read a Technical Article», *IEEE Software*, Nov./Dec. 1998, pp. 128f.

[86] McConnell, Steve, «After the Gold Rush», Redmond, WA: Microsoft Press, 1999.

[87] McCue, Gerald M., «IBM's Santa Teresa Laboratory – Architectural Design for Program Development», *IBM Systems Journal*, vol. 17, no. 1, 1978, pp. 4–25.

[88] McFarlan, F. W., «Portfolio Approach to Information Systems», *Harvard Business Review*, September-October 1981, pp. 142–150.

[89] Metzger, Philip W., «Managing a Programming Project», 2d Ed., Englewood Cliffs, NJ: Prentice Hall, 1981.

[90] Metzger, Philip W., and John Boddie, «Managing a Programming Project», 3d Ed., Upper Saddle River, NJ: Prentice Hall PTR, 1996.

[91] Mills, Harlan D., «Software Productivity», Boston, MA: Little, Brown, 1983.

[92] Moore, Geoffrey, «Crossing the Chasm», New York: Harper Business, 1991.

[93] Myers, Glenford J., «The Art of Software Testing», New York: John Wiley & Sons, 1979.

[94] NASA, «Manager's Handbook for Software Development, Revision 1», Document number SEL-84-101, Greenbelt, MD: Goddard Space Flight Center, NASA, 1990.

[95] NASA, «Recommended Approach to Software Development, Revision 3», Document number SEL-81-305, Greenbelt, MD: Goddard Space Flight Center, NASA, 1992.

[96] NASA Software Engineering Laboratory, «Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules», Document Number SEL-91-001, Greenbelt, MD: Goddard Space Flight Center, NASA, 1991.

[97] National Center for Education Statistics, «2001 Digest of Educational Statistics», Document Number NCES 2002130, April 2002.

[98] Neumann, Peter G., «Computer Related Risks», Reading, MA: Addison-Wesley, 1995.

[1] С. Макконнелл «Остаться в живых! Руководство для менеджера программных проектов». – Пер. с англ. – СПб.: Питер, 2006.

[99] Nuseibeh, Bashar, «Ariane 5: Who Dunnit?» *IEEE Software*, May/June 1997, pp. 15–16.

[100] «Occupational Outlook Handbook 2002-03 Edition», Bureau of Labor Statistics, 2002.

[101] Oldham, Leon G., et al., «Benefits Realized from Climbing the CMM Ladder», *Crosstalk*, May 1999.

[102] Olsen, Neil C., «Survival of the Fastest: Improving Service Velocity», *IEEE Software*, September 1995, pp. 28–38.

[103] Page-Jones, Meilir, «The Practical Guide to Structured Systems Design», Englewood Cliffs, NJ: Yourdon Press, 1988.

[104] Parnas, David L., «On the Criteria to Be Used in Decomposing Systems into Modules», *Communications of the ACM*, vol. 5, no. 12, December 1972, pp. 1053–58.

[105] Parnas, David L., «Designing Software for Ease of Extension and Contraction», *IEEE Transactions on Software Engineering*, v. SE-5, March 1979, pp. 128–138.

[106] Parnas, David, «On ICSE's „Most Influential" Papers», *Software Engineering Notes*, July 1995.

[107] Parnas, David L., «Software Engineering: An Unconsummated Marriage», *Software Engineering Notes*, November 1997.

[108] Parnas, David L., «Software Engineering Programmes Are Not Computer Science Programmes», *IEEE Software*, November/December 1999.

[109] Parnas, David Lorge, «Licensing Software Engineers in Canada», *Communications of the ACM*, November 2002, pp. 96–98.

[110] Pitterman, Bill, «Telcordia Technologies: The Journey to High Maturity», *IEEE Software*, July 2000.

[111] Pressman, Roger S., «Software Engineering: A Practitioner's Approach», 5th Ed., New York: McGraw-Hill, 2001.

[112] «Process Maturity Profile of the Software Community 2001 Year End Update», Software Engineering Institute, March 2002.

[113] Raghavan, Sridhar A., and Donald R. Chand, «Diffusing Software-Engineering Methods», *IEEE Software*, July 1989, pp. 81–90.

[114] Randall, Richard L., et al., «Product-Line Reuse Delivers a System for One-Fifth the Cost in One-Half the Time», *Crosstalk*, August 1996.

[115] Raymond, E.S., «Homesteading the Noosphere», 1998, *www.catb.org/~esr/writings/homesteading*.

[116] Reich, Charles, «The Greening of America», New York: Random House, 1970.

[117] Reifer, Donald J., «Making the Software Business Case: Improvement by the Numbers», Boston, MA: Addison-Wesley, 2001.

[118] Rich, Charles, and Richard C. Waters, «Automatic Programming: Myths and Prospects», *IEEE Computer*, August 1988.

[119] Rogers, Everett M., «Diffusion of Innovations», 4th Ed., New York: The Free Press, 1995.

[120] Rush, Gary, «The Fast Way to Define System Requirements», In Depth, *Computerworld*, October 7, 1985.

[121] Sackman, H., W. J. Erikson, and E. E. Grant, «Exploratory Experimental Studies Comparing Online and Offline Programming Performance», *Communications of the ACM*, vol. 11, no. 1, January 1968, pp. 3–11.

[122] «Scaling Up: A Research Agenda for Software Engineering», *Communications of the ACM*, March 1990.

[123] «Selections from Ralph Waldo Emerson», Edited by Stephen E. Whicher. Boston, MA: Houghton Mifflin Company, 1960.

[124] Shaw, Mary, «Prospects for an Engineering Discipline of Software», *IEEE Software*, November 1990, pp. 15ff.

[125] Simon, Herbert, «The Sciences of the Artificial», 3d Ed., Cambridge, MA: MIT Press, 1996.

[126] «Software-Engineering Methods», *IEEE Software*, July 1989, pp. 81–90.

[127] Sommerville, Ian, «Software Engineering», 6th Ed., Boston, MA: Addison-Wesley, 2000.

[128] Stallman, R.M., «The GNU Manifesto», 1985, *www.fsf.org/gnu/manifesto.html*.

[129] «Table 2, Employment by occupation, 1996 and projected 2006», in «Occupational projections to 2006», *Monthly Labor Review*, November 1997.

[130] «Table 255. – Bachelor's degrees conferred by degree-granting institutions, by discipline division: 1970–71 to 1999–2000», National Center for Education Statistics, *2001 Digest of Educational Statistics*, Document Number NCES 2002130, April 2002.

[131] Tackett, Buford D., III, and Buddy Van Doren, «Process Control for Error Free Software: A Software Success Story», *IEEE Software*, May 1999.

[132] The Standish Group, «Charting the Seas of Information Technology», Dennis, MA: The Standish Group, 1994.

[133] Thomsett, Rob, «Effective Project Teams: A Dilemma, a Model, a Solution», *American Programmer*, July–August 1990, pp. 25–35.

[134] Tripp, Leonard, «Professionalism of Software Engineering: Next Steps», Keynote Address at *12th Conference on Software Engineering Education and Training*, March 22, 1999.

[135] Twain, Mark, «Fenimore Cooper's Literary Offenses», 1895.

[136] van Solingen, Rini, «The Cost and Benefits of Software Process Improvement», *Proceedings of the Eighth European Conference on Information Technology Evaluation, September 17–18, 2001*.

[137] van Vliet, Hans, «Software Engineering Principles and Practice», West Sussex, England: John Wiley & Sons Ltd, 1993.

[138] Vosburgh, J., B. Curtis, R. Wolverton, B. Albert, H. Malec, S. Hoben, and Y. Liu, «Productivity Factors and Programming Environments», *Proceedings of the 7th International Conference on Software Engineering*, Los Alamitos, CA: IEEE Computer Society, 1984, pp. 143–152.

[139] Waligora, Sharon R., Linda C. Landis, Jerry T. Doland, «Closing the Loop on Improvement: Packaging Experience in the Software Engineering Laboratory», *Proceedings of the Nineteenth Annual Software Engineering Workshop, November 30–December 1, 1994*, NASA Goddard Space Flight Center, Greenbelt, MD, Document Number SEL-94-006.

[140] Wheeler, David, Bill Brykczynski, and Reginald Meeson, «Software Inspection: An Industry Best Practice», Los Alamitos, CA: IEEE Computer Society Press, 1996.

[141] White, John, and Barbara Simons, «ACM's Position on Licensing of Software Engineers», *Communications of the ACM*, November 2002, p. 91.

[142] Wiener, Lauren Ruth, *Digital Woes: Why We Should Not Depend on Software*, Reading, MA: Addison-Wesley, 1993.

[143] Wirth, Niklaus, «Algorithms + Data Structures», Englewood Cliffs, NJ: Prentice-Hall, 1985.[1]

[144] Yourdon, Edward and Larry L. Constantine, «Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design», Englewood Cliffs, NJ: Yourdon Press, 1979.

[145] Yourdon, Edward, «Rise and Resurrection of the American Programmer», Englewood Cliffs, NJ: Prentice Hall, 1996.

[146] Zachary, Pascal, «Showstopper! The Breakneck Race to Create Windows NT and the Next Generation at Microsoft», New York: Free Press, 1994.

[1] Н. Вирт «Алгоритмы и структуры данных», СПб.: Невский диалект, 2005.

Алфавитный указатель

Б

В

Г

Д

З

И

Й

К

Л

М

Н

Р

С

Т

У

Ф

Х

Ц

Ч

Ш

Э