ПРОФЕССИОНАЛЬНО

Фредерик БРУКС

как создаются программные системы

2-е ИЗДАНИЕ МЕЖДУНАРОДНОГО БЕСТСЕЛЛЕРА

Мифический человеко-месяц

ADDISON WESLEY

СИМВОЛ

Об авторе

Фредерик П. Брукс Младший является профессором вычислительной техники в школе бизнеса Кенан Университета штата Северная Каролина в Чэпел Хилл. Он известен, прежде всего, как «отец IBM System/360» и был руководителем проекта ее разработки, а позднее являлся руководителем программного проекта разработки операционной системы Operating System/360 на стадии проектирования. За эту работу Фредерик Брукс (Frederick Brooks), Боб Эванс (Bob Evans) и Эрик Блох (Erich Bloch) были в 1985 году награждены Национальной медалью в области технологии. До того Ф. Брукс разрабатывал в IBM архитектуру компьютеров Stretch и Harvest.

Доктор Брукс основал в Чэпел Хилл факультет вычислительной техники и возглавлял его с 1964 по 1984 годы. Он был членом Национального комитета по науке и Военного комитета по науке. В настоящее время он занимается преподаванием и исследованиями в области архитектуры компьютеров, молекулярной графики и виртуальных сред.

The Mythical Man-Month

Essays on Software Engineering
Anniversary Edition

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

ADDISON-WESLEY
An imprint of Addison Wesley Longman, Inc.

ПРОФЕССИОНАЛЬНО

Мифический человеко-месяц

или как создаются программные системы

Фредерик БРУКС

СИМВОЛ®

Санкт-Петербург — Москва
2007

Серия «Профессионально»

Фредерик Брукс

Мифический человеко-месяц или как создаются программные системы

Перевод С. Маккавеева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Редактор	*Н. Макарова*
Художник	*С. Борин*
Верстка	*С. Широкий*
Корректор	*С. Беляева*

ББК 32.973
УДК 681.3.06
Брукс Ф.
Б 89 Мифический человеко-месяц или как создаются программные системы. – Пер. с англ. – СПб.: Символ-Плюс, 2007. – 304 с.: ил.
ISBN 5-93286-005-7

Эта книга – юбилейное (дополненное и исправленное) издание своего рода библии для разработчиков программного обеспечения во всем мире, написанное Бруксом еще в 1975 году. Тогда же книга была издана на русском языке и давно уже стала библиографической редкостью. В США полагают, что без прочтения книги Брукса не может состояться ни один крупный руководитель программного проекта.

ISBN-13: 978-5-93286-005-2
ISBN-10: 5-93286-005-7
ISBN 0-201-83595-9 (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 12.07.2007. Формат 70 × 90 1/16. Печать офсетная.
Объем 19 печ. л. Доп. тираж 1000 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Посвящение издания 1975 года

Посвящается двоим людям, благодаря которым мои годы в IBM были особенно насыщенными:

Томасу Дж. Уотсону Младшему,
чье глубокое внимание к людям по-прежнему ощущается в его фирме,

и Бобу О. Эвансу,
чье смелое руководство превратило работу в приключение.

Посвящение издания 1995 года

Посвящается Нэнси,
Божьему дару для меня.

Содержание

Предисловие к изданию 1995 года

К моему удивлению и удовольствию, «Мифический человеко-месяц» остается популярным через 20 лет после выхода. Тираж превысил 250 000 экземпляров. Меня часто спрашивают, какие из оценок и рекомендаций, изложенных в 1975 году, я по-прежнему считаю верными, а какие претерпели изменения и в чем именно. Несмотря на то что в моих лекциях этот вопрос время от времени затрагивается, я давно жду возможности изложить его в печатном виде.

Питер Гордон (Peter Gordon), являющийся сейчас совладельцем издательства Addison-Wesley, терпеливо и с пользой сотрудничает со мной с 1980 года. Он предложил подготовить юбилейное издание. Мы решили не исправлять оригинал, а перепечатать его в неприкосновенности, за исключением обычных опечаток, и дополнить мыслями, возникшими в более позднее время.

В главе 16 перепечатывается статья «Серебряной пули нет: сущность и акциденция в программной инженерии», опубликованная IFIPS (Международная федерация по обработке информации) в 1986 году и явившаяся результатом опыта, полученного мной во время руководства исследованием применения программного обеспечения в военных областях, проводившимся Военным комитетом по науке. Мои соавторы по этому исследованию, а также наш исполнительный секретарь Роберт Л. Патрик оказали неоценимое содействие в моем возвращении к крупным практическим программным проектам. Статья была перепечатана в издании IEEE «Computer» в 1987 году, благодаря которому получила широкую известность.

Статья «Серебряной пули нет» была дерзкой. В ней предрекалось, что в течение ближайшего десятилетия не возникнет методов программирования, использование которых позволит на порядок величин повысить производительность разработки про-

граммного обеспечения при прочих равных условиях. До конца этого десятилетия остался год, и похоже, мое предсказание сбылось. Статья вызвала более оживленную дискуссию в печати, чем «Мифический человеко-месяц», поэтому в главе 17 содержатся ответы на некоторые из опубликованных критических замечаний, а также уточняются взгляды, изложенные в 1986 году.

При подготовке ретроспективного анализа и уточнения книги «Мифический человеко-месяц» я был удивлен тем, как мало содержавшихся в ней заявлений подверглось критике — как из числа доказанных, так и опровергнутых последующим опытом и исследованиями в области разработки программного обеспечения. Мне показалось полезным систематизировать эти заявления в чистом виде, без сопутствующих доказательств и данных. Я включил в книгу этот очерк в качестве главы 18, надеясь, что эти чистые утверждения вызовут поиск аргументов и фактов для доказательства, опровержения, пересмотра или уточнения.

Глава 19 собственно и представляет собой попытку пересмотреть изначальные утверждения. Следует предупредить читателя, что излагаемые новые взгляды далеко не в той мере подкреплены «боевым опытом», как это было в первой части книги. Дело в том, что в последнее время я работал в университетской среде, а не в промышленности, и над небольшими, а не крупномасштабными проектами. С 1986 года я занимаюсь только преподавательской деятельностью в области разработки программного обеспечения, но не исследованиями в ней. Моя исследовательская работа больше касается виртуальных сред и их применений.

При подготовке данной ретроспективы я поинтересовался современными взглядами своих друзей, которые занимаются разработкой программного обеспечения. В число тех, перед кем я в долгу за готовность поделиться своими взглядами, сделать полезные замечания к первоначальному тексту и усовершенствовать мое образование, входят Барри Бём (Barry Boehm), Кен Брукс (Ken Brooks), Дик Кейс (Dick Case), Джеймс Коггинс (James Coggins), Том Демарко (Tom DeMarco), Джим Мак-Карти (Jim McCarthy), Дэвид Парнас (David Parnas), Эрл Уилер (Earl Wheeler) и Эдвард Йордон (Edward Yordon). Фэй Уард (Fay Ward) прекрасно выполнила техническую работу, связанную с изданием новых глав.

Я благодарен моим коллегам из Группы по программному обеспечению для военных целей Военного комитета по науке: Гордону Беллу (Gordon Bell), Брюсу Бьюкенену (Bruce Buchanan), Рику Хейз-Роту (Rick Hayes-Roth) и особенно Дэвиду Парнасу — за их плодотворные идеи, а Ребеке Бирли (Rebekah Bierly) — за

подготовку к печати статьи, опубликованной в данной книге в качестве главы 16. Анализ проблем программирования в категориях «сущность» (essence) и «акциденция» (accident) возникло благодаря Нэнси Гринвуд Брукс, использовавшей такой анализ в статье, посвященной обучению игре на скрипке методом Сузуки.

Обычаи издательства Addison-Wesley не позволили мне в предисловии к изданию 1975 года выразить благодарность его сотрудникам за сыгранную ими важную роль. Следует особенно отметить вклад двух человек: ответственного редактора Нормана Стентона (Norman Stanton) и художественного редактора Герберта Боуза (Herbert Boes). Боуз создал изящный стиль, особо отмеченный одним из рецензентов: «широкие поля и творческое использование шрифтов и компоновки материала». И что еще важнее, он дал важный совет поместить в начале каждой главы свою картинку. (В то время у меня были только картинки Смоляных ям и Реймского собора.) Чтобы найти все картинки, мне потребовался целый год, но я бесконечно благодарен за совет.

Soli Deo gloria — Богу единому слава!

Чапел Хилл, Северная Каролина F. P. B., Jr.
Март 1995

Предисловие к первому изданию

Во многих отношениях управление большим проектом разработки программного обеспечения аналогично любому другому крупному начинанию — в большей мере, чем обычно считают программисты. Однако во многих отношениях имеются и отличия — в большей мере, чем обычно предполагают профессиональные менеджеры.

Идет процесс накопления профессиональных знаний в этой области. Состоялось несколько конференций, заседаний на конференциях AFIPS, опубликовано несколько книг и статей. Но знания еще не оформились в том виде, когда их можно систематически изложить в учебнике. Тем не менее представляется уместным предложить эту небольшую по объему книгу, отражающую, в основном, мои личные взгляды.

Мое профессиональное становление в вычислительной технике первоначально было связано с программированием, однако в период 1956–1963 годов, когда разрабатывались автономные управляющие программы и языки высокого уровня, я занимался в основном архитектурой компьютеров. Когда в 1964 году я стал менеджером проекта разработки Operating System/360, то обнаружил, что мир программирования совершенно изменился благодаря успехам, достигнутым за несколько последних лет.

Руководство разработкой OS/360 было очень поучительным, хотя и полным расстройств. Команде разработчиков, в том числе сменившему меня Ф. М. Трапнеллу (F. M. Trapnell), можно многим гордиться. Система содержит много отличных решений в конструкции и функционировании, и ей удалось получить широкое распространение. Некоторые идеи, в первую очередь организация ввода/вывода, независимая от устройств, и управление внешними библиотеками стали техническими новинками, ныне широко используемыми. Сейчас эта система вполне надежна, достаточно производительна и весьма гибка.

Однако проект нельзя назвать вполне успешным. Всякому пользователю OS/360 быстро становится ясно, насколько лучше могла бы быть система. Ошибки проектирования и реализации особенно заметны в управляющей программе, а не в компиляторах с языков. Большая часть этих просчетов относится к периоду разработки 1964–65 годов и потому должна быть отнесена на мой счет. Более того, система вышла с задержкой, потребовала больше памяти, чем предполагалось, стоимость разработки в несколько раз превысила запланированную и первые несколько версий функционировали не слишком удачно.

Покинув в 1965 году IBM и придя в Чэпел Хилл, как это и предполагалось, я возглавил разработку OS/360 и стал анализировать опыт этой разработки, чтобы извлечь уроки технологических решений и администрирования. В частности, я хотел понять, почему столь различным оказался опыт администрирования при разработке аппаратной части System/360 с одной стороны и создании операционной системы OS/360 — с другой. Эта книга является запоздалым ответом на вопросы Тома Уотсона относительно трудностей управления разработкой программ.

В решении этой задачи я получил большую пользу от длительного общения с Р. П. Кейсом (R. P. Case), помощником менеджера проекта в 1964–65 годах, и Ф. М. Тарпнеллом, менеджером проекта в 1965–68 годах. Я обсудил свои выводы с менеджерами других крупных программных проектов, в том числе Ф. Дж. Корбато (F. J. Corbato) из МТИ, Джоном Харром (John Harr) и В. Высоцким (V. Vyssotsky) из Bell Telephone Laboratories, Чарльзом Портманом (Charles Portman) из International Computers Limited, А. П. Ершовым из Вычислительного центра Сибирского отделения Академии наук СССР, а также с А. М. Пьетрасанта (A. M. Pietrasanta) из IBM.

Собственные мои выводы содержатся в следующих ниже очерках, предназначенных профессиональным программистам, профессиональным менеджерам и особенно профессиональным менеджерам в программировании.

Хотя книга написана в виде отдельных очерков, у нее есть центральная тема, излагаемая в основном в главах 2–7. Вкратце мое мнение заключается в том, что трудности, испытываемые при управлении крупными программными проектами, иного рода, нежели при управлении небольшими проектами, что связано с проблемами разделения труда. Я считаю важнейшей задачей сохранение концептуальной целостности самого продукта. В этих главах обсуждаются трудности, возникающие на пути к этому единству, и способы их преодоления. В главах, следующих за ними, рассмат-

риваются другие аспекты управления разработкой программного обеспечения.

Имеющаяся по этой теме литература не слишком богата, но весьма распылена. Поэтому я постарался включить ссылки на литературу, которые помогут осветить отдельные вопросы и отошлют заинтересованного читателя к другим полезным работам. Рукопись книги прочли многие мои друзья, и некоторые из них сделали пространные и полезные замечания. В тех случаях, когда, несмотря на ценность, они не вполне вписывались в текст, я включал их в примечания.

Поскольку эта книга представляет собой сборник очерков, а не единый текст, все ссылки и примечания вынесены в конец, и читателю при первом чтении можно их пропустить.

Я глубоко признателен мисс Саре Элизабет Мур (Sara Elizabeth Moore), мистеру Дэвиду Вагнеру (David Wagner) и миссис Ребекке Беррис (Rebecca Burris) за помощь в подготовке данной рукописи, а также профессору Джозефу Слоуну (Joseph C. Sloane) за советы в отношении иллюстраций.

Чэпел Хилл, Северная Каролина F. P. B., Jr.
Октябрь 1974

Глава 2

Этот мифический «человеко-месяц»

Чтобы приготовить вкусную пищу, требуется время. Если вам пришлось ждать, то лишь потому, что мы хотим лучше обслужить вас и доставить вам удовольствие.

МЕНЮ РЕСТОРАНА «АНТУАН» В НЬЮ-ОРЛЕАНЕ

Программные проекты чаще проваливаются из-за нехватки календарного времени, чем по всем остальным причинам вместе взятым. Почему эта причина неудач столь распространена?

Во-первых, слабо развиты наши методы оценок. В сущности, они отражают молчаливое и совершенно неверное предположение, что все будет идти хорошо.

Во-вторых, наши методы оценки ошибочно путают достигнутый прогресс с затраченными усилиями, неявно допуская, что скорость выполнения проекта пропорциональна количеству занятых в нем сотрудников.

В-третьих, поскольку менеджеры программных проектов неуверены в своих оценках, им часто недостает вежливого упрямства, как у шеф-повара ресторана «Антуан».

В-четвертых, выполнение графика работ слабо контролируется. Типовые, опробованные в других инженерных дисциплинах методы считаются радикальными нововведениями при разработке программного обеспечения.

В-пятых, при обнаружении отставания от графика естественной и общепринятой реакцией является увеличение числа разработчиков. Это все равно, что тушить пламя бензином. В результате дела идут значительно хуже. Чем сильнее пламя, тем больше нужно бензина, и в итоге этот путь приводит к катастрофе.

Контроль выполнения графика будет предметом отдельного разговора. Рассмотрим более подробно остальные аспекты проблемы.

Оптимизм

Все программисты — оптимисты. Возможно, эта современная разновидность колдовства особенно привлекательна для тех, кто верит в хэппи-энды и добрых фей. Возможно, сотни неудач отталкивают всех, кроме тех, кто привык сосредоточиваться на конечной цели. А может быть, дело всего лишь в том, что компьютеры и программисты молоды, а молодости свойственен оптимизм. Как бы то ни было, в результате имеем одно и то же: «На этот раз она точно пойдет!» Или: «Я только что выявил последнюю ошибку!»

Итак, в основе планирования разработки программ лежит ложное допущение, что *все будет хорошо*, т. е. *каждая задача займет столько времени, сколько «должна» занять*.

Глубокий оптимизм программистов заслуживает более серьезного изучения. Дороти Сэйерс (Dorothy Sayers) в своей превос-

ходной книге «The Mind of the Maker» (Разум творца) выделяет в творческой деятельности три стадии: замысел, реализацию, взаимодействие. Соответственно, книга, компьютер или программа сначала возникают как идеальное построение, существующее не во времени и пространстве, а лишь в мозгу своего создателя. Реализация же во времени и пространстве происходит с помощью пера, чернил, бумаги либо — проводов, кремния и феррита. Творение будет завершено, когда кто-либо прочтет книгу, воспользуется компьютером или запустит программу, тем самым вступив во взаимодействие с разумом их создателя.

Это описание, используемое Сэйерс для освещения не только творческой деятельности человека, но и христианского догмата Троицы, поможет нам в нашей текущей задаче. Для человека, который что-то создает, неполнота и противоречивость идей выявляются только при их реализации. Поэтому для теоретика изложение на бумаге, экспериментирование, изготовление являются неотъемлемыми частями творческого процесса.

Во многих видах творческой деятельности материал с трудом поддается обработке. Дерево колется, краски пачкаются, электрические цепи «звенят». Эти физические ограничения сужают круг идей, которые могут быть выражены, а также создают неожиданные трудности при реализации.

Реализация, таким образом, требует сил и времени как из-за физического материала, так и ввиду неадекватности основополагающих идей. Большую часть затруднений при реализации мы склонны объяснять недостатками физического материала, поскольку он «чужд» нам — в отличие от идей, которыми мы гордимся.

При создании же программ мы имеем дело с чрезмерно податливым материалом. Программист осуществляет свои построения на основе чистого мышления — понятий и очень гибких их представлений. Поскольку материал столь податлив, мы не ожидаем трудностей при реализации, отсюда и наш глубокий оптимизм. Из-за ошибочности наших идей возникают ошибки в программах. Следовательно, наш оптимизм не имеет оправдания.

Для отдельной задачи допущение, что все будет хорошо, оказывает на график работ вероятностный эффект. Все может действительно идти по плану, поскольку есть некоторое распределение вероятности для возможной задержки и существует конечная вероятность того, что задержки не будет. Однако большой программный проект состоит из множества задач, часть из которых может быть начата только после окончания других. Вероятность того, что все задачи будут завершены в срок, бесконечно мала.

Человеко-месяц

Вторая ошибка рассуждений заключена в самой единице измерения, используемой при оценивании и планировании: человеко-месяц. Стоимость действительно измеряется как произведение числа занятых на количество затраченных месяцев. Но не достигнутый результат. Поэтому *использование человеко-месяца как единицы измерения объема работы является опасным заблуждением*.

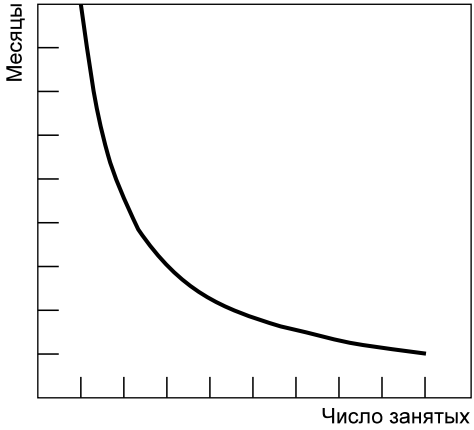

Рис. 2.1. Зависимость времени от числа занятых — полностью разделимая задача

Число занятых и число месяцев являются взаимозаменяемыми величинами лишь тогда, когда задачу можно распределить среди ряда работников, которые *не имеют между собой взаимосвязи* (рис. 2.1). Это верно, когда жнут пшеницу или собирают хлопок, но в системном программировании это далеко не так.

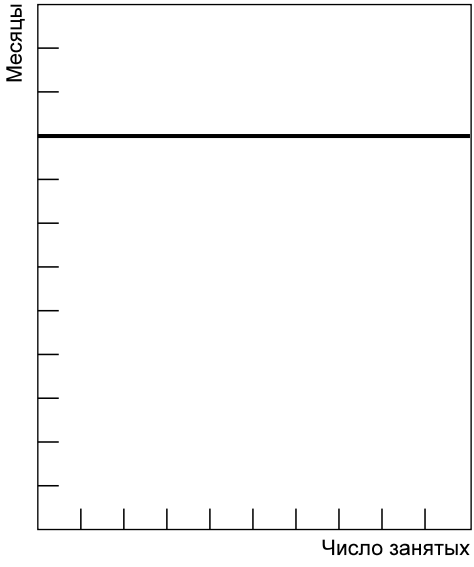

Рис. 2.2. Зависимость времени от числа занятых — неразделимая задача

Если задачу нельзя разбить на части, поскольку существуют ограничения на последовательность выполнения этапов, то увеличение затрат не оказывает влияния на график (рис. 2.2). Чтобы родить ребенка, требуется девять месяцев, независимо от того, сколько женщин привлечено к решению этой задачи. Многие задачи программирования относятся к этому типу, поскольку отладка по своей сути носит последовательный характер.

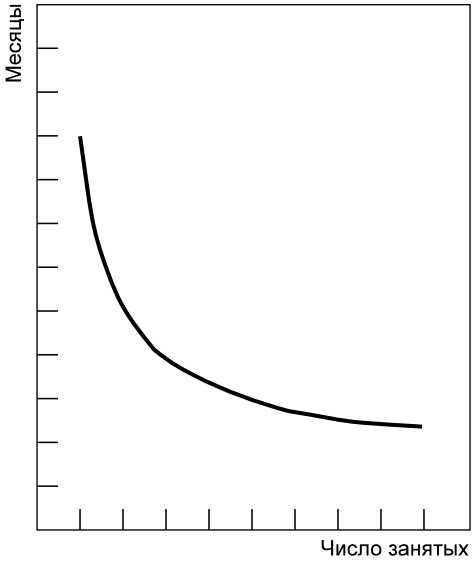

Рис. 2.3. Зависимость времени от числа занятых — разделимая задача, требующая обмена данными

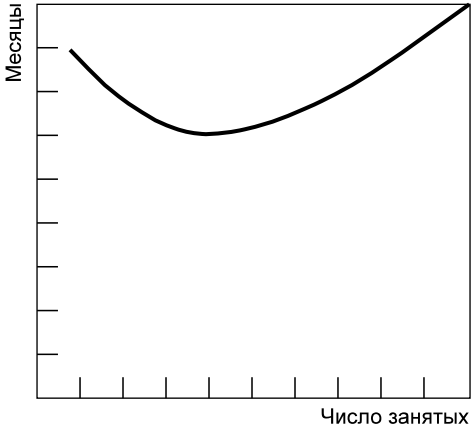

Рис. 2.4. Зависимость времени от числа занятых — задача со сложными взаимосвязями

Для задач, которые могут быть разбиты на части, но требуют обмена данными между подзадачами, затраты на обмен данными должны быть добавлены к общему объему необходимых работ. Поэтому достижимый наилучший результат оказывается несколько хуже, чем простое соответствие числа занятых и количества месяцев (рис. 2.3).

Дополнительная нагрузка состоит из двух частей — обучения и обмена данными. Каждого работника нужно обучить технологии, целям проекта, общей стратегии и плану работы. Это обучение нельзя разбить на части, поэтому данная часть затрат изменяется линейно в зависимости от числа занятых.

С обменом данными дело обстоит хуже. Если все части задания должны быть отдельно скоординированы между собой, то затраты возрастают как $n(n-2)/2$. Для трех работников требуется втрое больше попарного общения, чем для двух, для четырех — вшестеро. Если помимо этого возникает необходимость в совещаниях трех, четырех и т. д. работников для совместного решения вопросов, положение становится еще хуже. Дополнительные затраты на обмен данными могут полностью обесценить результат дробления исходной задачи и привести к положению, описываемому рис. 2.4.

Поскольку создание программного продукта является по сути системным проектом — практикой сложных взаимосвязей, затраты на обмен данными велики и быстро начинают преобладать над сокращением сроков, достигаемым в результате разбиения задачи на более мелкие подзадачи. В этом случае привлечение дополнительных работников не сокращает, а удлиняет график работ.

Системное тестирование

Из всех элементов графика работ наибольшему воздействию со стороны ограничений на последовательность выполнения действий подвержены отладка компонентов и системное тестирование. Кроме того, затраты времени зависят от количества выявленных ошибок и того, насколько они «скрытые». Теоретически ошибок быть не должно. Из-за своего оптимизма мы обычно склонны недооценивать действительное количество ошибок. Поэтому в программировании придерживаться графиков работ обычно труднее всего при отладке.

В течение ряда лет при планировании разработки программного обеспечения я пользуюсь следующим эмпирическим правилом:

$^1/_3$ — планирование,
$^1/_6$ — написание программ,
$^1/_4$ — тестирование компонентов и предварительное системное тестирование,
$^1/_4$ — системное тестирование при наличии всех компонентов.

Это правило имеет несколько важных различий с общепринятым планированием:

1. На планирование отводится больше времени, чем обычно. И все равно этого времени едва достаточно для разработки подробных и надежных технических условий и недостаточно для проведения исследовательских работ или поиска новейших технологий.
2. *Половина* графика работ, отведенная на отладку законченного кода, значительно выше нормы.
3. Та часть, которую легко оценить, т. е. написание кода, занимает всего одну шестую общего времени.

Изучая проекты, график которых был составлен традиционным образом, я обнаружил, что немногие из них отводили по графику половину времени на отладку, но на практике в боль-

шинстве случаев тратили на нее половину фактического времени. Многие проекты укладывались в график на всех этапах, исключая системное тестирование.[2]

Особенно катастрофические последствия может иметь недостаток времени для системного тестирования. Поскольку задержка происходит в конечной части графика, никто не подозревает о том, что график находится под угрозой срыва вплоть до дня сдачи продукта. Плохие вести, полученные поздно и без предупреждения, обескураживают клиентов и менеджеров.

Более того, задержка на этом этапе имеет особенно тяжелые материальные и психологические последствия. Проект осуществляется при полной укомплектованности работниками и максимальных ежедневных финансовых издержках. Что важнее, программное обеспечение должно обеспечить поддержку другой деловой активности (поставки компьютеров, запуска новых производственных мощностей и т. п.), и связанные с задержкой вторичные издержки очень высоки. На практике эти вторичные издержки могут быть выше, чем все прочие. Поэтому очень важно в изначальном графике работ отвести достаточно времени для системного тестирования.

Робость в оценках

Для программиста, как и для повара, давление со стороны хозяина может определять запланированный срок завершения задачи, но не может определять время ее фактического завершения. Омлет, обещанный через две минуты, может успешно жариться, но если через две минуты он не готов, то у клиента есть две возможности: ждать еще или съесть его сырым. Тот же выбор встает и перед заказчиком программного обеспечения.

У повара есть еще одна возможность: добавить жару. В результате омлет часто оказывается безнадежно испорченным: горелым с одного края и сырым — с другого.

Я не думаю, что у менеджеров программных проектов меньше храбрости или твердости, чем у поваров или других менеджеров в инженерных областях. Но липовые графики, нацеленные на желательную хозяину дату, встречаются здесь значительно чаще, чем в любых других инженерных областях. Очень тяжело, рискуя потерять рабочее место, с энергией и любезностью отстаивать срок, который определен без применения каких-либо количественных методов при недостатке данных и подкреплен, в основном, интуицией менеджера.

Очевидно, необходимо сделать две вещи. Мы должны получить и сделать общедоступными численные данные, характеризующие производительность, частоту программных ошибок, методы оценки и т. д. Вся отрасль может только выиграть от опубликования таких данных.

Пока методы оценивания не получат более прочной основы, менеджерам остается только мужаться и защищать свои прогнозы, утверждая, что полагаться на их слабую интуицию все же лучше, чем основываться на одних желаниях.

Действия при срыве графика

Что делают, когда важный программный проект начинает отставать от графика? Естественно, добавляют людей. Как показывают рис. 2.1–2.4, это не всегда помогает.

Рассмотрим пример.[3] Предположим, что трудоемкость задачи оценивается в 12 человеко-месяцев, и три человека должны выполнить ее за 4 месяца, причем в конце каждого месяца имеются четыре контрольные точки A, B, C, D, в которых можно произвести измерения (рис. 2.5).

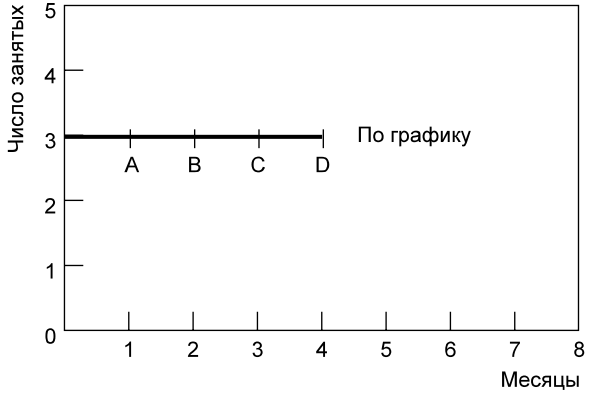

Рис. 2.5

Предположим теперь, что первая контрольная точка была достигнута лишь по истечении двух месяцев. Какие альтернативы имеются у менеджера?

1. Допустим, что необходимо соблюсти срок выполнения задачи, и ошибочно оценена была только первая часть задачи, т. е. рис. 2.6 верно отражает положение. Значит, остается 9 челове-

ко-месяцев трудозатрат и два месяца, поэтому понадобится $4^1/_2$ человека, и к троим имеющимся нужно добавить еще двоих.

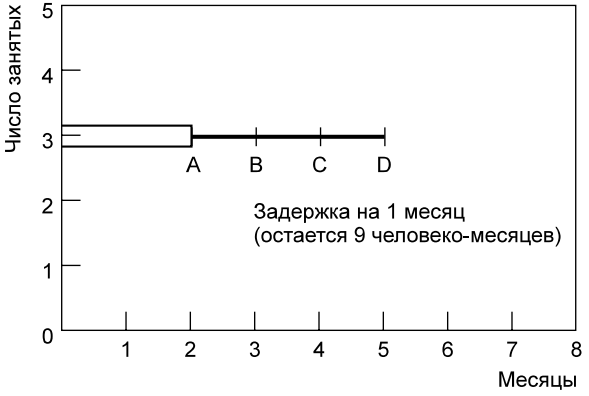

Рис. 2.6

2. Допустим, что необходимо соблюсти срок выполнения задачи, и одинаково занижена была вся оценка, т. е. положение соответствует рис. 2.7. Значит, остается 18 человеко-месяцев трудозатрат и два месяца, поэтому понадобится 9 человек. К троим имеющимся нужно добавить еще шестерых.

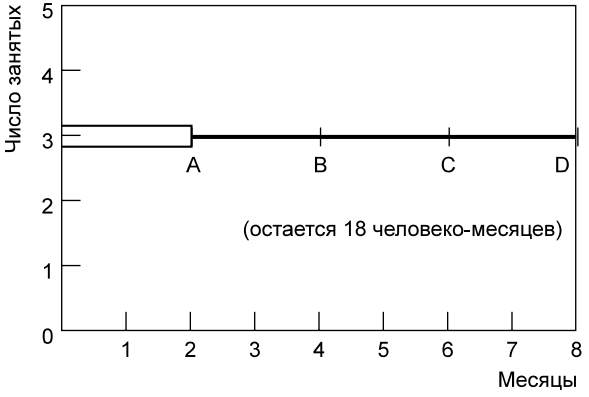

Рис. 2.7

3. Изменить график. Мне нравится замечание, сделанное П. Фаггом (P. Fagg), опытным инженером по вычислительной технике: «Маленьких задержек не бывает». Это означает, что в новом графике должно быть достаточно времени, чтобы работа была исполнена тщательно и полностью и не пришлось бы вновь переделывать график.

4. Сократить задачу. На практике этим всегда и кончается, когда команда обнаруживает, что не укладывается в график. Когда очень высоки вторичные издержки, это единственное, что можно сделать. Менеджеру предоставляется возможность официально и аккуратно сократить задачу, изменить график, либо наблюдать, как задача молча урезается при поспешном изменении проекта и неполном тестировании.

В первых двух случаях настаивать на том, чтобы задача в неизменном виде была выполнена за четыре месяца, чревато катастрофой. Рассмотрим, к примеру, восстановительный эффект первой альтернативы (рис. 2.8). Двое новых работников, какими бы знающими они ни были и как бы быстро не удалось их найти, должны изучить задачу с помощью одного из опытных разработчиков. Если для этого потребуется месяц, то *3 человеко-месяца будут потрачены на работу, которая не учитывается в исходной оценке*. Кроме того, задача, разбитая первоначально на три потока, теперь должна быть перекроена на пять частей. Поэтому часть уже сделанной работы будет потеряна, а системное тестирование нужно будет продлить. В результате в конце третьего месяца останется работы существенно больше, чем на 7 человеко-месяцев, а в распоряжении будет 5 подготовленных человек и один месяц. Согласно рис. 2.8 продукт будет запаздывать так же, как если бы ни одного человека не было добавлено (см. рис. 2.6).

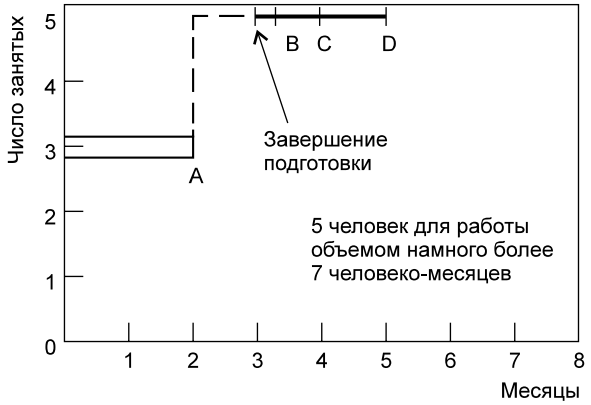

Рис. 2.8

Если рассчитывать управиться за четыре месяца с учетом только времени обучения, но не перераспределения задач и дополнительного системного тестирования, то в конце второго месяца потребуется добавить 4, а не 2 человек. Чтобы компенсировать воздействие перераспределения задач и системного тестирования,

потребуются еще новые люди. Теперь, однако, команда состоит не из 3, а по крайней мере, 7 человек, и такие вопросы, как организация команды и распределение задач, приобретают новый качественный уровень.

Обратите внимание, что к концу третьего месяца дело выглядит весьма мрачно. Несмотря на все административные усилия контрольная точка, намеченная на 1 марта, не достигнута. Возникает сильный соблазн повторить цикл, добавив еще людей. Это безумное решение.

В предшествующих рассуждениях предполагалось, что только первая контрольная точка была неверно рассчитана. Если 1 марта сделать консервативное предположение, что весь график был излишне оптимистичен, как отражено на рис. 2.7, требуется добавить 6 человек к исходной задаче. Расчет воздействия обучения, перераспределения задач и системного тестирования предоставляется сделать читателю в качестве упражнения. Нет сомнений, что при попытке уложиться в срок в итоге получится худший продукт, чем при изменении графика и сохранении первоначальных троих человек без усиления.

Крайне упрощая, сформулируем Закон Брукса:

Если проект не укладывается в сроки, то добавление рабочей силы задержит его еще больше.

Это развенчивает миф о человеко-месяце. Продолжительность осуществления проекта зависит от ограничений, накладываемых последовательностью работ. Максимальное количество разработчиков зависит от числа независимых подзадач. Эти две величины позволяют получить график работ, в котором будет меньше занятых разработчиков и больше месяцев. (Единственная опасность заключается в возможном устаревании продукта.) Нельзя, однако, составить работающие графики, в которых занято больше людей и требуется меньше времени. Программные проекты чаще проваливаются из-за нехватки календарного времени, чем по всем остальным причинам вместе взятым.