9 Время и затраты

Самый быстрый способ реализации любой системы — построение ее по критическому пути. Хорошо спроектированный проект также эффективно распределяет минимально необходимые ресурсы по критическому пути, но продолжительность реализации проекта по-прежнему ограничивается его критическим путем. Выполнение можно ускорить за счет применения инженерных практик, обеспечивающих быструю и чистую разработку. Кроме этих практик разработки, в этой главе рассматриваются возможности сокращения сроков разработки за счет сжатия критического пути. Основной способ сокращения сроков — переработка проекта с определением нескольких более коротких планов проекта с еще большей степенью сжатия. После этого проявляется фундаментальная концепция кривой «время-затраты» и взаимодействие времени и затрат в проекте. В результате вы получаете набор вариантов планирования проекта, которые позволяют приспособиться к изначальным пожеланиям руководства относительно времени и затрат, а также быстро адаптироваться к возможным непредвиденным изменениям.

Ускорение программных проектов

Вопреки распространенному мнению, для соблюдения сроков недостаточно просто больше работать или назначить в проект больше людей. Для этого нужно работать умнее, четче и правильнее, применяя профессиональный опыт. В общем случае в любом программном проекте для ускорения проекта в целом могут применяться следующие методы:

- *Обеспечение качества*. В большинстве команд операции контроля качества и тестирования ошибочно причисляются к области обеспечения качества, или QA (Quality Assurance). Истинная сфера QA не имеет отношения к тестированию. Как правило, в ней задействуется опытный эксперт, который отвечает на вопрос: что потребуется для обеспечения качества? В его ответе должно быть указано, как ориентировать весь процесс разработки для

контроля качества, как предотвратить само возникновение проблем, как обнаружить и исправить корневые причины проблем. Само присутствие QA-специалиста — признак организационной зрелости, который почти всегда свидетельствует о приверженности качеству, понимании того, что качество не образуется само по себе, и подтверждении того, что организация должна активно к нему стремиться. QA-специалист иногда отвечает за планирование процесса и проработку ключевых фаз. Поскольку качество ведет к продуктивности, правильные меры обеспечения качества всегда ускоряют работу по графику и выделяют организации, практикующие QA, из остальных представителей отрасли.

- *Привлечение инженеров по тестированию.* Инженеры по тестированию — не рядовые тестировщики, а полноценные программисты, которые проектируют и пишут код, предназначенный для нарушения работоспособности кода системы. Как правило, инженеры по тестированию обладают более высокой технической квалификацией, чем обычные разработчики, потому что написание тестового кода часто включает более трудные задачи: разработку фиктивных каналов коммуникаций; проектирование и разработку регрессионного тестирования; проектирование тестовых оснасток, имитаторов, средств автоматизации и т. д. Инженеры по тестированию до мелочей знают архитектуру и внутреннее устройство системы и пользуются этими знаниями для того, чтобы нарушать работу системы на каждом шаге. Наличие такой «антисистемной» системы, готовой развалить ваш продукт, творит чудеса для качества, потому что вы можете выявлять проблемы сразу же после их возникновения, изолировать корневые причины, избегать каскадных эффектов при изменениях, устранять суперпозиции дефектов, маскирующих другие дефекты, и значительно сокращать цикл решения проблем. Ничто не ускоряет работу по графику в такой степени, как постоянная, свободная от дефектов кодовая база.

- *Добавление тестировщиков.* Как правило, в командах разработчиков больше, чем тестировщиков. В проектах, в которых недостаточно тестировщиков, один-два тестировщика не могут угнаться за командой, и их участие часто ограничивается проведением тестирования с незначительной полезностью. Такое тестирование слишком монотонно, оно не изменяется в зависимости от размера команды или растущей сложности системы, а система часто рассматривается как «черный ящик». Это не означает, что качественное тестирование при этом не проводится, это означает, что скорее его основная часть перекладывается на разработчиков. Изменение пропорции тестировщиков к разработчикам до 1:1 или даже 2:1 (в пользу тестировщиков) позволяет разработчикам проводить меньше времени за тестированием и больше — за созданием непосредственной пользы для проекта.

- *Вложения в инфраструктуру.* Всем программным системам требуются такие общие средства, как средства безопасности, очереди сообщений и шина сообщений, размещение, публикация событий, ведение журнала, инстру-

ментальные средства, диагностика и профилирование, а также регрессионное тестирование и автоматизация тестирования. Многим программным системам требуются средства управления конфигурацией, сценарии развертывания, процесс сборки, ежедневные сборки и тесты состояния (все это часто объединяется в категорию DevOps). Вместо того чтобы заставлять каждого разработчика писать собственную уникальную инфраструктуру, следует вложиться в построение (и сопровождение) инфраструктурного каркаса для всей команды, который будет решать большинство перечисленных задач. Это позволяет разработчикам сосредоточиться на задачах программирования, имеющих прямую коммерческую ценность, обеспечивает экономию за счет масштаба, помогает новым разработчикам быстрее войти в курс дела, снижает уровень стресса и трения в команде, а также сокращает время разработки системы.

- *Повышение квалификации разработки*. Для современных сред характерен очень высокий темп изменений. Многие разработчики не могут угнаться за новейшими языками, инструментами, фреймворками, облачными платформами и другими новшествами. Даже самые лучшие разработчики вечно осваивают новые технологии, и они проводят непропорциональное время за неструктурированными, хаотичными поисками. Что еще хуже, некоторые разработчики настолько не справляются с нагрузкой, что занимаются копированием кода из интернета без реального понимания краткосрочных и долгосрочных последствий (включая юридические) своих действий. Чтобы справиться с этой проблемой, следует выделять время и ресурсы для обучения разработчиков технологиям, методологиям и имеющимся инструментам. Наличие компетентных разработчиков ускорит разработку любого программного продукта.

- *Совершенствование процесса*. Многие среды разработки страдают от ограниченности процесса. Они выполняют все положенные действия как ритуал, но не имеют четкого понимания или правильного восприятия, лежащего в основе активностей. Активности, лишенные реального содержания, не приносят никакой реальной пользы и часто только ухудшают ситуацию по принципу карго-культа[1]. О процессах разработки программных продуктов были написаны целые тома. Изучайте проверенные передовые практики и разработайте план совершенствования, который решит проблемы качества, сроков и бюджета. Отсортируйте методы, входящие в план совершенствования, по результативности и простоте принятия и заранее анализируйте причины, по которым они не применялись ранее. Сформулируйте стандартные оперативные процедуры, настаивайте на том, чтобы команда и вы сами следовали этим процедурам, и даже заставляйте при необходимости. Со временем ваши проекты станут более предсказуемыми, и вы сможете реализовывать их в соответствии с заданным графиком.

[1] https://ru.wikipedia.org/wiki/Карго-культ

- *Адаптация и применение стандартов.* В подробном стандарте программирования определяются правила формирования имен и стиль, практики программирования, настройки и структура проекта, ваши собственные правила, регламенты вашей команды и известные потенциальные проблемы. Стандарт помогает обеспечить применение передовых практик разработки и избегать ошибок, чтобы новички могли быстро подняться до уровня ветеранов. Код становится более единообразным, и пропадают те проблемы, которые часто встречаются при работе над кодом, написанным другим разработчиком. Соблюдение стандартов повышает вероятность успеха и сокращает время, которое могло бы уйти на разработку системы.
- *Доступность для внутренних экспертов.* В большинстве команд вы не найдете экспертов мирового уровня. Задача команды — понять суть бизнеса и создать систему, а не продемонстрировать свою квалификацию в безопасности, размещении, UX, облачных технологиях, AI, BI, больших данных или архитектуре баз данных. На эти попытки «изобретения велосипеда» тратится много времени, и они никогда не бывают такими же эффективными, как использование доступных и проверенных знаний (вспомните проблему 2% из главы 2). Гораздо лучше и быстрее довериться внутренним экспертам. Используйте этих экспертов там, где это потребуется, — это позволит избежать дорогостоящих ошибок.
- *Проведение партнерского рецензирования.* Лучший отладчик — человеческий глаз. Разработчики часто обнаруживают проблемы в коде коллег намного быстрее, чем потребуется для диагностики и исправления проблем после того, как код станет частью системы. Этот принцип также справедлив для дефектов требований, для плана проектирования и плана тестирования для каждого из сервисов в системе. Команда должна проанализировать все эти аспекты, чтобы обеспечить наивысшее качество кодовой базы.

> **ПРИМЕЧАНИЕ** Отрасль разработки программного обеспечения настолько хаотична, что эти практики на уровне здравого смысла могут показаться чужеродными многим современным разработчикам. Тем не менее если вы проигнорируете их и будете делать то же, что и раньше, только в большем объеме, это не приведет к ускорению проекта. Действия, породившие проблему, не могут использоваться для ее решения. Со временем применение этих практик (или хотя бы их части) повысит производительность вашей команды, ее приверженность успеху и способность уверенно браться за сложные планы.

Передовые практики программирования ускоряют проект в целом независимо от конкретных активностей или сетевого графика самого проекта. Они эффективны в любом проекте, в любой среде и при использовании любых технологий. Хотя такой способ улучшения проекта может показаться дорогостоящим, в конечном итоге он может привести к экономии. Сокращение времени, затраченного на разработку системы, компенсирует дополнительные затраты на ее улучшение.

Уплотнение графика

К сожалению, ни один из элементов приведенного выше списка методов ускорения не гарантирует быстрого решения проблем с графиком; всем им требуется время, чтобы проявить свою эффективность. Тем не менее существует пара возможностей для немедленного ускорения работы — вы должны либо работать с лучшими ресурсами, либо поискать возможности выполнения параллельной работы. Применение этих приемов позволяет вам уплотнить график проекта. Уплотнение графика вовсе не означает, что та же работа будет выполняться быстрее. Речь идет о том, что вы будете достигать тех же целей быстрее, причем часто для более быстрого завершения задачи или проекта придется выполнить больший объем работы. Эти два способа уплотнения могут использоваться в сочетании друг с другом или по отдельности, в отдельных частях проекта, в проекте в целом или на уровне отдельных активностей. Оба способа уплотнения в конечном итоге повышают прямые затраты проекта (определение приводится далее) при сокращении сроков.

Использование лучших ресурсов

Старшие разработчики реализуют свою часть системы быстрее, чем младшие. Тем не менее существует распространенное заблуждение, будто это различие объясняется тем, что они быстрее программируют. Часто младшие разработчики программируют намного быстрее, чем старшие. Старшие разработчики проводят за программированием минимально возможную часть своего времени, а основное время проводится за проектированием программного модуля, взаимодействий и методов, которые они собираются использовать при тестировании. Старшие разработчики пишут тестовые оснастки, имитаторы и эмуляторы для компонентов, над которыми они работают, и для потребляемых ими сервисов. Они документируют свою работу, предвидят последствия каждого решения относительно программирования, учитывают удобство сопровождения и расширяемость своих сервисов, а также такие аспекты, как безопасность. Следовательно, хотя на единицу времени старшие разработчики пишут код медленнее, чем младшие, они быстрее справляются со своими задачами. Как нетрудно догадаться, старшие разработчики пользуются высоким спросом и запрашивают более высокую оплату, чем младшие. Эти лучшие ресурсы должны назначаться только на критические активности, поскольку их использование за пределами критического пути не повлияет на график.

Параллельная работа

Как правило, если вы берете последовательный набор активностей и находите возможность выполнять эти активности параллельно, это приводит к сокращению графика. Существуют два возможных способа параллельной работы.

В первом случае вы извлекаете внутренние фазы активности и перемещаете их в другую точку проекта. Во втором случае вы удаляете зависимости между активностями, чтобы работать над этими активностями параллельно (как объяснялось в главе 7, одновременное назначение нескольких людей на одну активность не работает).

Расщепление активностей

Вместо того чтобы выполнять внутренние фазы активности последовательно, можно расщепить активность. Некоторые из менее зависимых фаз планируются параллельно с другими активностями проекта, до или после активности. Среди хороших кандидатов для внутренних фаз, перемещаемых к началу проекта (то есть до остатка активности), можно выделить подробное проектирование, документирование, построение эмуляторов, план тестирования сервисов, тестовую оснастку, проектирование API, UI-проектирование и т. д. Кандидаты для внутренних фаз, перемещаемых к концу проекта, — интеграция с другими сервисами, модульное тестирование и повторное документирование. Расщепление активности сокращает время, занимаемое ею на критическом пути, и сокращает продолжительность работы над проектом.

Удаление зависимостей

Вместо того чтобы последовательно работать над зависимыми активностями, стоит поискать возможности сокращения и даже устранения зависимостей между активностями и организации параллельной работы над ними. Если в проекте присутствует активность A, зависящая от активности B, которая, в свою очередь, зависит от активности C, продолжительность проекта будет определяться суммой продолжительностей этих трех активностей. Но если вам удастся удалить зависимость между A и B, вы сможете работать над A параллельно с B и C, что приведет к соответствующему уплотнению графика.

Удаление зависимостей часто требует вложений в дополнительные активности, которые делают возможной параллельную работу:

- *Проектирование контракта.* Создав отдельную активность для проектирования контракта сервиса, вы можете предоставить интерфейс или контракт потребителям, а потом начать работать над ними еще до завершения сервиса, от которого они зависят. Возможно, предоставление контракта не снимет зависимость полностью, но откроет возможность для выполнения параллельной работы до определенного уровня. То же относится к проектированию UI, сообщений, API и протоколов между подсистемами и даже системами.
- *Разработка эмуляторов.* Для спроектированного контракта можно написать простой сервис, который эмулирует реальный сервис. Такая реализация может быть очень простой (например, всегда возвращать одни и те же

результаты без ошибок), что приведет к дальнейшему устранению зависимостей.

- *Разработка имитаторов.* Вместо простого эмулятора можно разработать полноценный имитатор сервиса или группы сервисов. Имитатор может поддерживать состояние, внедрять ошибки и обладать поведением, неотличимым от поведения реального сервиса. Иногда написание хорошего имитатора может быть делом более сложным, чем построение реального сервиса. Тем не менее имитатор разрывает зависимости между сервисом и его клиентами, позволяя реализовать более высокую степень параллелизма в работе.
- *Повторная интеграция и тестирование.* Даже если у вас есть отличный имитатор для сервиса, клиент, разработанный для этого имитатора, должен быть причиной для беспокойства. После того как реальный сервис будет завершен, вы должны повторить интеграцию и тестирование этого сервиса со всеми клиентами, разработанными для имитатора.

Кандидаты для параллельной работы

Иногда лучшие кандидаты для параллельной работы очевидны из диаграммы распределения кадров. Если диаграмма содержит несколько импульсов, возможно, эти импульсы можно разделить.

Взгляните на диаграмму на рис. 9.1: на ней хорошо видны три импульса. В исходном плане все три выполнялись последовательно из-за зависимостей между выходом каждого импульса, который становился входом для следующего импульса. Если вам удастся каким-то образом удалить эти зависимости, вы сможете работать над одним или двумя импульсами параллельно, что приведет к значительному уплотнению графика.

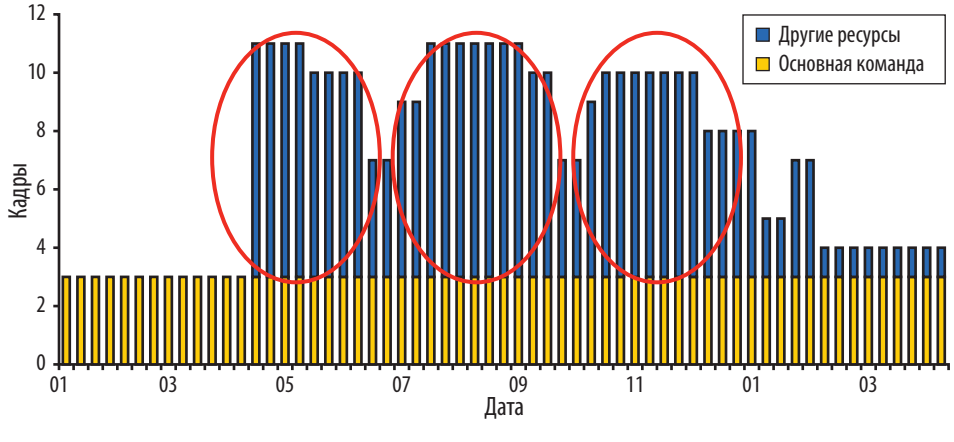

Рис. 9.1. Кандидаты для параллельной работы

Параллельная работа и затраты

Обе формы параллельной работы — расщепление активностей и удаление зависимостей между активностями — часто требуют дополнительных ресурсов. Для выполнения извлеченных фаз параллельно с другими активностями проекту часто требуется больше ресурсов. Проекту также потребуются дополнительные ресурсы для работы над дополнительными активностями, обеспечивающими параллельную работу, например дополнительными разработчиками для повторной интеграции и дополнительными тестировщиками для повторного тестирования. Это повышает стоимость проекта и рабочую нагрузку. В частности, дополнительные ресурсы приводят к увеличению размера команды, увеличению пикового размера команды, повышению шума и меньшей эффективности выполнения. Снижение эффективности обернется еще большим повышением затрат, потому что вы будете получать меньше от каждого участника команды.

Существующая команда может быть не способна к параллельной работе по разным причинам (отсутствие архитектора, отсутствие старших разработчиков, неподходящий размер команды), и вам придется пользоваться услугами дорогостоящих внешних специалистов. Даже если вы можете себе позволить общую стоимость проекта, параллельная работа повысит скорость оборота денежных средств, а проект может стать неприемлемым. Короче говоря, параллельная работа не бесплатна.

Опасности параллельной работы

Обычно удаление зависимостей между активностями сродни обезвреживанию мины — действовать нужно очень осторожно. Параллельная работа часто увеличивает сложность исполнения проекта, а это заметно повышает требования к менеджеру проекта, ответственному за проект. Прежде чем браться за параллельную работу, стоит заняться инфраструктурой, которая бы ускоряла все активности в проекте без изменения зависимостей между активностями. Пожалуй, это проще и безопаснее параллельной работы.

Несмотря на все сказанное, параллельная работа сократит время выхода на рынок. Выбирая решение об уплотненном параллельном режиме, тщательно взвесьте риски и затраты на параллельное выполнение с ожидаемым сокращением графика.

Кривая зависимости затрат от времени

По крайней мере изначально повышение затрат позволяет быстрее завершить работу над любым проектом. В большинстве проектов зависимость между временем и затратами нелинейна, но в идеале она напоминает кривую на рис. 9.2.

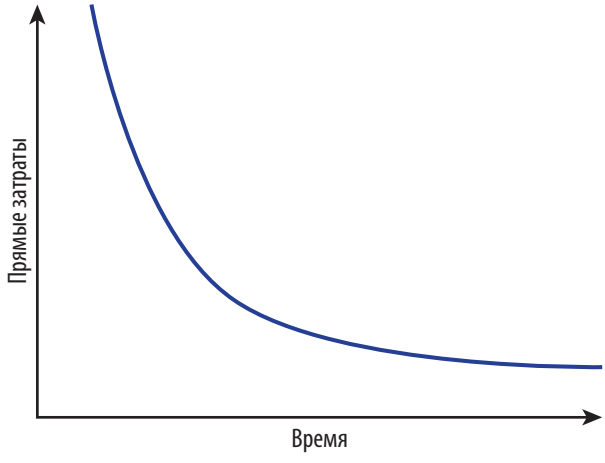

Рис. 9.2. Идеальная кривая «время-затраты»

Например, представьте проект на 10 человеко-лет, который состоит исключительно из активностей, связанных с программированием. Если поручить этот проект одному разработчику, на его выполнение уйдет 10 лет. С другой стороны, с двумя разработчиками тот же проект с большой вероятностью займет 7 и более лет, а не 5. Чтобы завершить проект за 5 лет, с большой вероятностью потребуются как минимум 3 разработчика, а скорее 5 или даже 6. Эти затраты (10 лет для 10 человеко-лет затрат, 7 лет для 14 человеко-лет, 5 лет для 30 человеко-лет) действительно выражают нелинейную зависимость затрат от времени.

Точки на кривой «время-затраты»

Кривая «время-затраты», изображенная на рис. 9.2, идеальна и нереалистична. Она предполагает, что при достаточно большом бюджете проект может быть реализован практически мгновенно. Здравый смысл подсказывает, что это предположение ошибочно. Например, ни при каком финансировании не удастся завершить проект на 10 человеко-месяцев за месяц (или хотя бы за год). У любых усилий по уплотнению графика есть естественный предел. Аналогичным образом кривая «время-затраты» на рис. 9.2 показывает, что при увеличении времени затраты на проект уменьшаются, тогда как выделение проектам времени большего, чем реально необходимо, приводит к повышению их затрат (как обсуждалось в главе 7).

Хотя кривая «время-затраты» на рис. 9.2 неверна, мы можем обсудить некоторые ее аспекты, встречающиеся во всех проектах. Эти аспекты являются результатом нескольких классических предположений из области планирования. На рис. 9.3 изображена реальная кривая «время-затраты».

Рис. 9.3. Реальная кривая «время-затраты» [адаптировано по материалам James M. Antill and Ronald W. Woodhead, Critical Path in Construction Practice, 4th ed. (Wiley, 1990)]

Нормальное решение

Проект всегда можно спроектировать в предположении о том, что вы располагаете неограниченными ресурсами и что каждый ресурс будет доступен тогда, когда он потребуется. В то же время проект следует планировать с учетом минимальных затрат и по возможности стараться не запрашивать больше ресурсов, чем действительно необходимо. Как объяснялось в главе 7, вы можете вычислить наименьший уровень ресурсов, который позволит беспрепятственно перемещаться по критическому пути. Вы получите наименее затратный способ построения системы и формирования самой эффективной команды. Такой вариант плана проекта называется *нормальным решением*. Нормальное решение представляет наименее ограниченный, или *естественный*, способ построения системы.

Неэкономичная зона

Предположим, продолжительность нормального решения проекта составляет один год. Если на этот же проект выделяется более одного года, он всегда будет стоить дороже. Дополнительные затраты происходят от блокировки ресурсов на более длительные периоды времени, от накапливаемых лишних затрат, от украшательства, от повышения сложности, а также от сокращения вероятно-

сти успеха. Таким образом, точки справа от нормального решения на кривой «время-затраты» относятся к неэкономичной зоне проекта.

Уплотненные решения

Нормальное решение можно уплотнить при помощи методов, описанных ранее в этой главе. Хотя все полученные уплотненные решения имеют меньшую продолжительность, они также обходятся дороже — скорее всего, в нелинейной зависимости.

Очевидно, все усилия по уплотнению должны быть направлены только на активности на критическом пути, потому что уплотнение некритических активностей никак не отражается на графике. Все уплотненные решения располагаются слева от нормального решения на кривой «время-затраты».

Минимальная продолжительность решения

По мере сжатия проекта затраты растут. В какой-то момент критический путь будет полностью уплотнен, потому что других кандидатов для параллельной работы нет, а вы уже назначили своих лучших людей на критические активности. При достижении этой точки вы получаете решение с минимальным временем (продолжительностью). Каждый проект всегда имеет точку минимальной продолжительности, в которой никакое количество денег, усилий или воли уже не ускорит его реализацию.

Решение с полным уплотнением

Невозможно построить проект за срок, меньший его наименьшей возможной продолжительности, но потерять деньги можно всегда. Ничто не мешает вам уплотнить все активности в проекте — как критические, так и некритические. Этот проект не будет завершен быстрее минимальной продолжительности, но безусловно обойдется дороже. Эта точка на кривой «время-затраты» называется *точкой полного уплотнения*.

> **ПРИМЕЧАНИЕ** Мой личный опыт показывает, что 30% с большой вероятностью является верхним пределом уплотнения по времени для любого программного проекта, причем даже этот уровень достигается с трудом. Вы можете воспользоваться этим порогом для проверки любых ограничений по срокам. Например, если проект имеет нормальное решение на 12 месяцев, а дедлайн установлен равным 7 месяцам, то этот проект построить не удастся, потому что он требует 41% уплотнения графика.

Дискретное моделирование

На реальной кривой «время-затраты», изображенной на рис. 9.3, между нормальным решением и решением минимальной продолжительности находит-

ся бесконечное количество точек. Конечно, никто не располагает временем для проектирования бесконечного количества решений, да и в этом нет необходимости. Вместо этого архитектор и менеджер проекта должны представить руководству одну-две точки между нормальным решением и решением минимальной продолжительности. Эти варианты представляют разумные сочетания времени и затрат, из которых руководство может выбрать наиболее подходящий вариант, и они всегда являются результатом некоторого уплотнения сети. В результате кривая, которая будет построена в ходе планирования проекта, является дискретной моделью (рис. 9.4). Хотя кривая «время-затраты» на рис. 9.4 содержит гораздо меньше точек, чем на рис. 9.3, она дает достаточно информации для того, чтобы правильно понять поведение проекта.

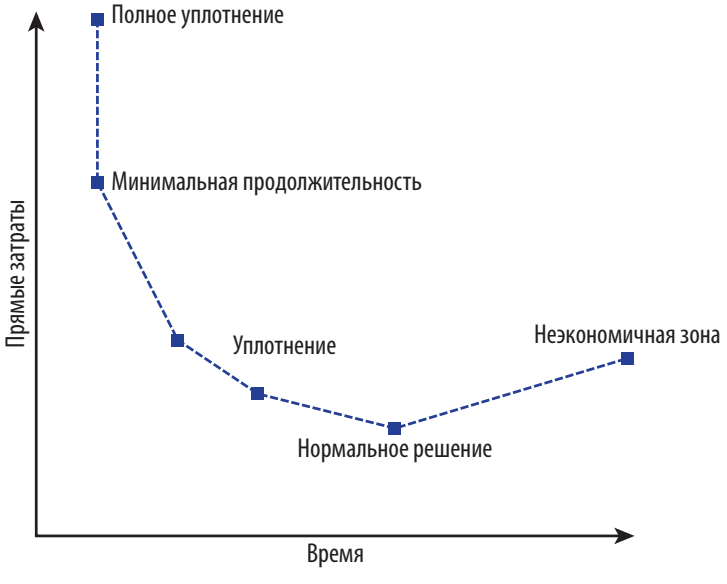

Рис. 9.4. Дискретная кривая «время-затраты» [адаптировано по материалам James M. Antill and Ronald W. Woodhead, Critical Path in Construction Practice, 4th ed. (Wiley, 1990)]

Предотвращение классических ошибок

Непрактичные решения с полным уплотнением и неэкономичные решения также стоит представить руководству, потому что многие руководители просто не знают об их непрактичности. У них вполне может быть ошибочная мысленная модель поведения проекта — скорее всего, сходная с изображенной на рис. 9.2. А с ошибочной мысленной моделью всегда принимаются ошибочные решения.

Допустим, график имеет наивысший приоритет, и руководитель готов на любые затраты ради соблюдения обязательств. Руководитель может думать, что выделение дополнительных средств и людей поможет команде продвинуться ближе к дедлайну, хотя никакие деньги не способны реализовать проект до минимальной продолжительности.

Также часто встречаются руководители с ограниченным бюджетом, но более свободным графиком. Такой руководитель может попытаться сократить затраты за счет субкритического комплектования проекта или непредоставления проекту необходимых ресурсов. При этом проект перемещается справа от нормального решения в неэкономичную зону, что приводит к значительному возрастанию затрат.

Реализуемость проекта

Кривая «время-затраты» отражает важнейший аспект проекта: его *реализуемость*. Планы проекта с сочетаниями времени и затрат, представляющими точки на кривой или над ней, могут быть воплощены в жизнь. Для примера возьмем точку A на рис. 9.5. Решение A требует времени T_2 и затрат C_1. Хотя решение A жизнеспособно, оно является субоптимальным. Если T_2 лежит в рамках допустимого дедлайна, то проект также может быть реализован с затратами C_2 — значение кривой «время-затраты» на момент времени T_2. Так как точка A лежит выше кривой, отсюда следует, что $C_2 < C_1$. И наоборот, если затраты C_1 приемлемы, для тех же затрат проект также может быть реализован за время T_1 — значение кривой «время-затраты» для времени C_1. Так как A находится справа от кривой, отсюда следует, что $T_1 < T_2$.

Точки на кривой «время-затраты» просто представляют оптимальное соотношение затрат и времени. Кривая «время-затраты» оптимальна, потому что всегда лучше реализовать проект быстрее (при тех же затратах) или с меньшими затратами (при том же сроке). Проект может быть реализован хуже, но не лучше кривой «время-затраты».

Также отсюда следует, что точки под кривой «время-затраты» невозможны. Например, возьмем точку B на рис. 9.6. Решение B требует времени T_3 и затрат C_4. Однако реализация проекта за время T_3 потребует затрат не менее C_3. Так как точка B находится под кривой «время-затраты», отсюда следует, что $C_3 > C_4$. Если вы не можете позволить себе затраты выше C_4, то проект потребует времени не менее T_4. Так как точка B находится слева от кривой «время-затраты», отсюда следует, что $T_4 > T_3$.

Мертвая зона

Если точки на кривой «время-затраты» представляют минимальные затраты для любой продолжительности, кривая «время-затраты» делит плоскость на две зоны. Первая — зона возможных решений — объединяет решения на кри-

вой «время-затраты» и выше нее. Вторая — мертвая зона — объединяет все решения ниже кривой «время-затраты» (рис. 9.7).

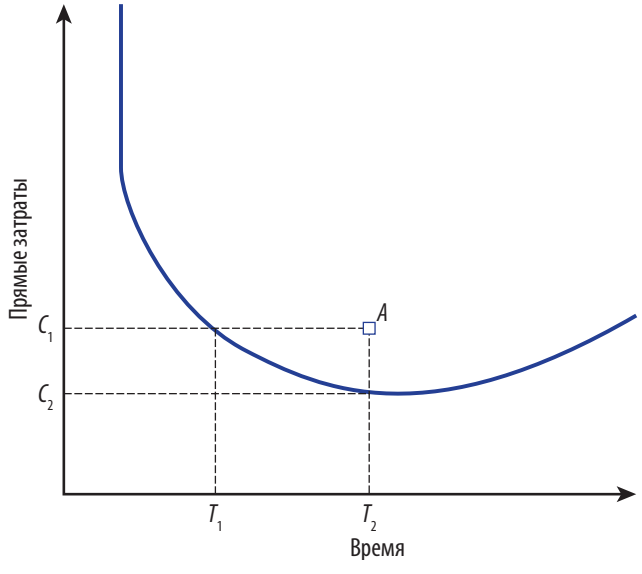

Рис. 9.5. Субоптимальное решение над кривой «время-затраты»

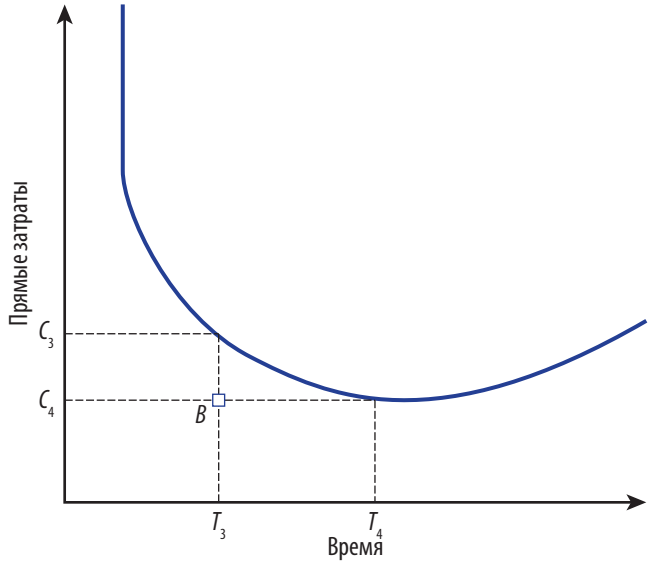

Рис. 9.6. Невозможное решение под кривой «время-затраты»