Содержание

ГЛАВА 4

Как добиться успеха при разработке мобильных приложений

Из словаря Merriam-Webster
Основная запись: ***meth· od· ol· o· gy (методология)***
Функция: существительное
Форма(ы) с окончаниями: мн. ***-gies***
Этимология: нов. лат. ***methodologia****, от лат.*
methodus *+* ***-logia -logy***
Дата: 1800
1: *Совокупность методов, правил и допущений, используемых в определенной дисциплине: конкретная процедура или набор процедур.*
2: *Анализ принципов и процедур, применяемых при проведении исследований в определенной области.*

(www.m-w.com, 2004)

Введение

Эта глава посвящена отдельному рассмотрению методологии разработки программного обеспечения, используемой в процессе проектирования и построения мобильных приложений.

Можно выделить два рода проблем, с которыми приходится сталкиваться разработчикам приложений, специалистам по архитектуре систем и руководителям при разработке программного обеспечения. С одной стороны, это трудности, действительно присущие разработке программного обеспечения как таковой, с другой — трудности временного характера, отражающие несовершенство инструментальных средств разработки и технологий, доступных на данном этапе.

Что касается разработки приложений для настольных компьютеров и серверов, то за последние 10–15 лет в разрешении "временных" проблем в этой области был достигнут огромный прогресс.

Современные средства разработки несравненно более производительны по сравнению с теми, которые использовались какой-нибудь десяток лет тому назад. Это особенно справедливо по отношению к проектированию и отладке пользовательских интерфейсов. Аналогичные преобразования в настоящее время происходят и в области разработки программного обеспечения для мобильных устройств. Многие усовершенствования, касающиеся проектирования и отладки программ для настольных компьютеров и серверов, сейчас доступны и для мобильных устройств, что значительно упрощает выполнение этих видов работ и делает их доступными для более широкого круга разработчиков, чем несколько лет тому назад. Темпы разработки программ резко возросли. В результате этого значительно увеличилось число проектов, которые могут быть реально осуществлены (за некоторые из них при других условиях не имело бы смысла даже и браться). Благодаря достижениям в области средств и технологий программирования сейчас вполне осуществимы такие проекты мобильного программного обеспечения, реализация которых еще несколько лет тому назад была невозможной из-за их высокой сложности и стоимости.

Однако, как и в случае разработки программ для настольных компьютеров и серверов, между "возможностью написания кода" и "возможностью создания замечательного приложения" лежит огромная пропасть. Реализация последней из указанных возможностей сопряжена с трудностями, присущими разработке любого современного программного обеспечения. Одно дело — просто взяться за работу, написать и отладить код, и совершенно другое — связать весь код в корректно функционирующее, надежное и гибкое приложение.

Путь, позволяющий преодолеть трудности, свойственные разработке программного обеспечения, пролегает через использование подходящих методологий. В двух словах, методология — это набор руководящих принципов, соблюдение которых позволяет успешно завершить начатое дело. Многие из методологий, ориентированных на приложения для настольных компьютеров и серверов, в равной степени можно использовать и при разработке программного обеспечения для мобильных устройств, однако трудности, которые являются специфическими для устройств или становятся в этом случае особенно заметными, заслуживают отдельного рассмотрения.

Данная глава разделена на две части. В первой из них обсуждается общая методология разработки программ с добавлением в необходимых случаях отдельных пояснений, относящихся к частному случаю мобильных устройств. Во второй части главы рассматриваются узкие методологические вопросы, связанные непосредственно с особенностями разработки мобильных приложений. Задача начальных разделов данной главы состоит в том, чтобы развить в вас понимание всей важности выбора подходящей методологии разработки, тогда как в остальных разделах настоящей главы раскрываются принципы конкретной методологии, которыми следует руководствоваться при разработке программного обеспечения для мобильных устройств.

Трудности постоянного и временного характера, с которыми приходится сталкиваться при разработке программного обеспечения

НА ЗАМЕТКУ

Данная тема великолепно исследуется и обсуждается в книге *The Mythical Man Month* (*Мифический человеко-месяц*) Фредерика Брукса (Frederick Brooks). Каждый, кто хочет получить солидную подготовку в области общей методологии разработки программного обеспечения, должен обязательно прочесть эту книгу.

Трудности временного характера и способы их преодоления

Некоторые из проблем, с которыми приходится сталкиваться разработчикам программного обеспечения, можно с полными основаниями охарактеризовать как "временные трудности". С появлением более совершенных инструментальных средств разработки острота этих проблем может снизиться. Хорошим примером временных трудностей могут служить проблемы отладки. За многие годы был достигнут огромный прогресс в предоставлении разработчикам средств отладки приложений в процессе их выполнения. Это привело к кардинальным изменениям самого процесса отладки, благодаря чему эта ранее трудоемкая задача, которая решалась при помощи столь разнородного инструментария, как карандаш и бумага, низкоуровневые средства наподобие дизассемблеров или отладочная печать, и требовала немалой доли интуиции, в настоящее время превратилась в весьма естественную интерактивную составную часть любого набора современных средств разработки. Среди разработчиков, пользующихся современным инструментарием, сегодня вряд ли найдутся такие, кто считает отладку отдельным видом деятельности, отличным от проектирования и написания кода. Сейчас все это объединено в рамках одного естественного процесса разработки программного обеспечения, и разработчики плавно переключаются с одного вида деятельности на другой. Еще не так давно наблюдалась явно иная картина, и, по крайней мере, в отношении программного обеспечения, выполняющегося на устройствах, об этом можно говорить со всей определенностью. Представляя собой когда-то сложнейшую проблему, отладка кода в наши дни значительно упростилась. То были временные трудности, и улучшение технологий позволило с ними справиться.

Трудности постоянного характера и методологии, привлекаемые для их разрешения

Ко второй категории проблем разработки программного обеспечения относятся те, которые лучше всего описываются выражением "неотъемлемые трудности". Проблемы этого рода уходят корнями в саму сердцевину процесса разработки программ.

Никакими усовершенствованиями инструментальных средства разработки решить эти проблемы невозможно. Скорее, эти проблемы диктуют необходимость применения подходящих *методологий*, которые могли бы направлять техническую мысль в нужное русло и гарантировать успешное завершение программных проектов, невзирая на трудности.

Неплохим примером ситуаций, которым свойственны неизбежные сложности, является проектирование алгоритмов. Современные объектно-ориентированные языки разработки значительно облегчили инкапсуляцию и организацию кода, но они не в состоянии упростить или автоматизировать собственно проектирование алгоритмов. Несомненно, наличие богатого набора базовых классов, предлагаемых современными программными средами, способствует написанию эффективных алгоритмов, однако проектирование принципиально новых алгоритмов по-прежнему является нелегкой задачей, которая, по всей вероятности, будет оставаться таковой и в ближайшем обозримом будущем. Это имеет место по той простой причине, что алгоритмам, в силу самой их природы, свойственна специфичность, обусловленная конкретикой задачи, и не существует никаких известных общих способов, позволяющих преобразовать ваши намерения в алгоритм, который наилучшим образом соответствовал бы намеченным целям; в подобных случаях автоматизация возможна лишь при условии значительного сужения масштаба задачи и создания специального инструментария для ее решения. То же самое можно сказать и о написании многопоточного кода. Применение усовершенствованных инструментальных средств, языков программирования и библиотек, несомненно, облегчает эту работу и позволяет решать задачи во многих случаях, допускающих строгое описание; вместе с тем, создать универсальную машину, способную разрез*а*ть общие задачи на параллельные ломтики, нам никак не удается. Подобные проблемы являются неизбежными, и их решение требует тщательного проектирования и применения подходящей методологии.

Современные языки программирования и средства графического проектирования позволяют разработчикам полнее реализовывать свои намерения, но не устраняют необходимости в приобретении устойчивых навыков конструирования алгоритмов. Без таких навыков невозможно обойтись при построении критических систем, определяющих поведение и эффективность программного обеспечения. Лучшее, что смогла предложить современная технология программирования, — это упаковка сложных алгоритмов в повторно используемые компоненты и каркасы, и предоставление возможности моделирования взаимодействия компонентов между собой. При таком подходе проектирование критических систем, представляющих общий интерес, могут осуществлять высококлассные специалисты, а остальные разработчики получают возможность повторно использовать эти системы. Проектирование компонентов может быть упрощено, а взаимодействие между авторами компонентов и клиентами сделано более прозрачным за счет применения таких технологий моделирования, как UML (Unified Modeling Language — унифицированный язык моделирования) и панели графического проектирования, но эти технологии не в состоянии снять проблемы внутренней сложности, свойственные процессу проектирования эффективных алгоритмов. Постоянное совершенствование инструментальных средств будет способствовать преодолению временных трудностей, однако трудности постоянного характера, обусловленные самой природой алгоритмов, будут всегда оставаться.

Компоненты находят чрезвычайно широкую применимость, так как позволяют повторно воспользоваться результатами однажды проделанной нелегкой работы, но

проектирование алгоритмов от этого нисколько не упрощается. Компонентно-ориентированное проектирование является *методологией*, возникшей из необходимости помочь программистам справиться с одной из самых неприятных проблем, с которыми приходится сталкиваться при разработке программного обеспечения. В соответствии с этой методологией разработчикам и специалистам в области архитектуры программных систем рекомендуется разбивать свои задачи на раздельные многоуровневые компоненты.

Сначала выделяются критические компоненты и алгоритмы, требующие проектирования, кодирования и тестирования на самом высоком профессиональном уровне. Вся основная функциональность предоставляется разрабатываемым приложениям высокоуровневым и, как правило, менее строго тестируемым кодом, который использует эти компоненты. Причина успеха компонентов как методологии заключается в том, что она дает возможность разложить приложение на отдельные части. Это позволяет инженерам-программистам, специалистам в области архитектуры систем и руководителям идентифицировать наиболее трудные моменты алгоритмизации и сосредоточить на них внимание. Как и в случае любой другой методологии, при разбиении приложения на компоненты необходимо действовать осмотрительно, стараясь не переусердствовать в этом. Если выделять в отдельные компоненты все, что только возможно, исходя из того ложного допущения, что увеличение количества компонент, на которые разбивается проект, приводит к лучшему техническому решению, то в результате этого чрезмерно усложнятся интерфейсы между многочисленными компонентами различного рода. Средства моделирования могут оказать вам помощь в визуализации этих взаимодействий, но от сложной картины вам все равно не избавиться. Чрезмерное обилие необязательных компонентов размывает тот острый фокус, который, в соответствии с самим ее назначением, должна обеспечивать разбивка на компоненты для достижения высоких эксплуатационных характеристик критических элементов. Любая методология должна применяться осмотрительно и с отчетливым пониманием тех целей, которые при этом преследуются. Методология полезна лишь тогда, когда можно измерить ее эффективность при решении данной задачи, ибо только в этом случае вы будете уверены в том, что реализуются все ее преимущества.

Две ситуации, благоприятные с точки зрения применения методологии разбивки на компоненты

Двумя областями, в которых использование компонентов предлагает эффективные способы решения сложных задач и значительно упрощает работу с этими технологиями, являются XML и криптография.

Синтаксический анализ XML-выражений является примером непростой технической задачи, для решения которой очень хорошо подходит методология повторного использования компонентов. Число разработчиков, самостоятельно создающих собственные XML-анализаторы для своих приложений, весьма невелико лишь по той простой причине, что спроектировать и реализовать высокопроизводительный XML-анализатор, отличающийся высокой степенью надежности и универсальностью, очень нелегко. Если бы все те, кому необходимо использовать в своих приложениях XML, должны были создавать собственные XML-анализаторы, то последние страдали бы всевозможными мелкими огрехами и неувязками. Это нанесло бы непоправимый ущерб самой возможности организации взаимодействия между различными платформами, лежащей в основе XML. Исключая случаи чисто академических упражнений в проектировании алгоритмов (а XML предоставляет для этого действительно благодатную почву), написание собст-

венного XML-анализатора было бы глупой затеей; вам никогда не удастся выполнить эту работу так же хорошо, как ее могут сделать те люди, перед которыми поставлена единственная задача: написать и протестировать такой анализатор. Вместо того чтобы заставлять каждого разработчика программного обеспечения тратить время на написание собственных специальных программ синтаксического разбора XML-выражений, методология проектирования, рекомендует применять готовые универсальные компоненты, прошедшие тщательное тестирование. В результате разработок, выполненных на чрезвычайно строгом уровне проектирования алгоритмов и тестирования, были созданы несколько XML-анализаторов, основанных как на собственном, так и на управляемом коде. Эти несколько компонентов повторно используются многими разработчиками приложений, желающими применять XML. В этом и состоит суть методологического подхода к решению изначально трудной задачи создания надежного XML-анализатора.

Другим замечательным примером применения методологии повторного использования компонентов, где она оказывается более эффективной по сравнению с пользовательскими вариантами реализации, может служить криптография. Проектирование криптографических алгоритмов является своего рода сложным искусством. Вступая в определенное противоречие с ярко выраженным аспектом своей специфичности, криптографические функциональные средства с каждым днем приобретают все большее значение для разработчиков, создающих самые рядовые приложения. Давно доказано — лучшим способом создания незащищенного приложения является написание собственных криптографических алгоритмов. Если только вашей целью не является проведение исследований в области криптографии, то вы должны знать, что устоявшаяся инженерная практика программирования категорически рекомендует не заниматься созданием собственных криптографических систем. Проектирование, реализация и сопровождение криптографической системы, отвечающей требованиям надежности, живучести и быстродействия, — работа не из легких. Было бы крайне неразумно и чревато многими отрицательными последствиями, если бы все, кто нуждается в функциональных средствах криптографии, самостоятельно создавали криптографические системы для своих приложений. Итак, здесь мы также сталкиваемся с неизбежными проблемами проектирования алгоритма, решение которых более совершенные инструменты облегчить для нас не в состоянии. Но вместо того, чтобы поднимать руки вверх, признавая свое поражение: “Да, это действительно трудная задача, и с этим ничего не поделаешь”, следует обратиться к методологии программных разработок, основанной на разбиении сложных алгоритмических задач общего характера на отдельные компоненты, а также к методологии, в соответствии с которой все участники рабочей группы должны следовать согласованному подходу, основанному на использовании компонентов.

Дополнительным преимуществом методологии, предполагающей использование готовых, берущихся “прямо с полки” компонент для решения проблем подобного рода, является снижение затрат на сопровождение программных продуктов. При обнаружении каких-либо изъянов в компонентах (а таковые всегда найдутся) они корректируются централизованным образом, а не по отдельности для каждого из многочисленных разрозненных вариантов реализации.

Любой достаточно сложный проект по разработке мобильного программного обеспечения нуждается в хорошем наборе методологий, способных справляться со свойственными данному виду разработки трудностями постоянного характера. Должен существовать процесс, позволяющий определять трудные проблемы и убеждаться в том, что они решаются наиболее подходящим для этого способом. Технологогия компонентов позволяет справиться с одним специфическим типом проблем разработки программного обеспечения. Однако для решения других проблем, например, тех, которые связаны с обеспечением необходимой производительности приложений, проектированием эффективных пользовательских интерфейсов или созданием надежных каналов связи для мобильных устройств, потребуются дополнительные мето-

дологии. Умение распознавать проблемы постоянного характера и знание способов их преодоления является отличительным признаком хорошего руководителя группы разработчиков.

Итак, существуют проблемы, в разрешении которых нам могут помочь инструментальные средства, и по мере совершенствования доступного инструментария эти проблемы будут год от года терять свою остроту. Наряду с этими проблемами временного характера существуют проблемы постоянного характера, которые могут решаться лишь на основе применения правильных технических подходов. Это возможно лишь при наличии методологии, гарантирующей, что наиболее важные проблемы будут решаться не от случая к случаю, а на основе их всестороннего и глубокого рассмотрения, как они того заслуживают.

Эволюция подходов к разработке программного обеспечения на протяжении ряда лет

Каждый успешный этап развития вычислительных средств характеризовался своими специфическими проблемами и соответствующими методологиями, позволяющими добиваться успеха. Эти методологии базировались и продолжают базироваться как на самых современных из имеющихся инструментальных средствах разработки, так и на специфических видах разрабатываемых решений.

На первых порах вычислительные ресурсы, например, память, регистры или тактовая частота процессоров были весьма ограниченными, и поэтому требование написания как можно более компактных и эффективных алгоритмов носило императивный характер. Эти цели являются актуальными и на сегодняшний день, но теперь они уравновешиваются долговременными преимуществами, которые сулит написание легких в обслуживании кодов, допускающих возможность их повторного использования. На данном этапе основной целью является не просто написание как можно более эффективного кода, а написание наиболее эффективного кода, отличающегося ясностью, надежностью и легкостью сопровождения.

Пакетная обработка

В эпоху пакетных вычислений (то есть во времена, предшествовавшие появлению диалогового программного обеспечения) безраздельно царствовали алгоритмы. Тогда было возможно, и это имело действительно существенное значение, указать как входные, так и ожидаемые выходные данные системы еще до написания кода. В силу того, что модель применения программных продуктов описывалась схемой ввод —> обработка —> вывод, много времени тратилось на разработку центрального алгоритма, а главной задача проектирования заключалась в максимальной экономии дискового пространства и памяти. У этой модели есть свои достоинства, и сегодня она по-прежнему является идеалом, к которому следует стремиться при разработке отдельных процедур, предназначенных для обработки информации в пакетном режиме (например, при проектировании алгоритмов сортировки). Такой подход целесообразно применять при проектировании отдельных функций, но не сложных систем, обеспечивающих интерактивное взаимодействие с пользователем. Этот период можно назвать эрой “блок-схем”.

Серверная обработка без сохранения состояний

При построении надежных серверных приложений, отвечающих на запросы, роль Святого Грааля играет “отсутствие состояния”. Ваш код получает запрос, обрабатывает его, а затем возвращает результат, не нуждаясь в сохранении состояния на протяжении периода времени между двумя последовательными запросами. Все это очень напоминает пакетную обработку, поскольку в данном случае также используется модель, описываемая схемой ввод —> обработка —> вывод. Разумеется, современные сложные Web-приложения (например, система покупательских тележек на Web-сайте Amazon) долж-

ны сохранять свое состояние в течение промежутков времени между двумя последовательными запросами, а этого легче всего достигнуть, осуществляя управление состояниями на основе существенно централизованного механизма инкапсуляции, обеспечивающего выполнение как можно большего объема кода без сохранения состояния.

Интерактивные вычисления, управляемые событиями

Управляемые событиями вычисления (event-driven computing) представляют вычислительную модель, в которой приложение находится в состоянии долговременного обмена информацией с конечным пользователем. Выполнение кода осуществляется в ответ на действия и запросы конечного пользователя; задача приложения заключается в обеспечении соответствующей реакции на то, что делает пользователь. В отличие от пакетных приложений интерактивные (диалоговые) приложения не имеют определенного фиксированного периода завершения и продолжают обрабатывать новые события, запускаемые пользователем, до тех пор, пока пользователь не решит прервать рабочий сеанс. Эту модель программирования обычно называют "управляемой событиями", поскольку в результате действий пользователя вырабатываются дискретные события (например, события щелчка мыши, выбора элемента списка или закрытия окна), на которые должно реагировать разрабатываемое приложение. При построении удачного интерактивного приложения, управляемого событиями, очень многое зависит от того, насколько тщательно продумана модель управления состояниями.

Разработка программ является итеративным процессом, который, тем не менее, также должен подчиняться определенным правилам

В наши дни лишь немногие специалисты рекомендуют использовать блок-схемы для представления функционального поведения приложений. Начнем с того, что в этом случае приходится работать с блок-схемами огромных размеров. Современная модель программ, управляемых событиями, богатые пользовательские интерфейсы и широкодоступные инструментальные средства, обеспечивающие быструю разработку приложений, привели к тому, что идея отображения всех алгоритмов приложения и вариантов пользовательского взаимодействия с помощью блок-схем в настоящее время кажется уже старомодной. Детализация на таком уровне может быть использована для отображения отдельных ключевых алгоритмов, если это необходимо, но в целом требуется более высокоуровневый подход.

Существует и противоположный подход, ориентированный на конкретную специфику ситуаций, суть которого состоит в том, что необходимо "просто взяться за работу и составлять код, пока не будет получено завершенное приложение". При всей своей соблазнительности такой бездумный подход к программированию чреват возможными просчетами и обычно приводит к созданию беспорядочного клубка плохо функционирующего кода с массой "ляпов", который трудно сопровождать и от которого, в конце концов, вы откажетесь и будете переписывать заново. Приложения этого типа часто выглядят так, будто их "склепали" или "слепили" лишь потому, что под рукой оказались их составные части. В этом случае все проектные решения принимаются в результате ответов на то и дело возникающие частные вопросы о том, каким образом можно решить ту или иную текущую проблему, и при этом мало внимания уделяется

анализу того, каково будет общее влияние данного решения на систему в целом как в отношении ее поведения, так и в отношении сложности кода.

В отличие от только что описанного подхода, отличающегося отсутствием систематичности, сторонники другого направления утверждают: "Прежде чем написать хотя бы одну строку кода, необходимо составить подробные спецификации для всего приложения в целом". Такой уровень детальной координации может потребоваться для проектов, осуществляемых группами разработчиков, но работа над такими проектами требует также постоянного контроля за актуальностью текущих спецификаций, чтобы они оставались действенными на протяжении всего цикла разработки. Кроме того, при работе с проектами подобного рода требуются иметь хорошее предварительное представление о том, каким должно быть конечное поведение приложения, что во многих случаях просто нереально. Как ни заманчиво звучит призыв "специфицировать все до мельчайших деталей", такой подход обычно оказывается непригодным для современных процессов разработки, и в частности, если речь идет о разработке первой версии продукта, поскольку по мере продвижения работы многое будет проясняться, и это повлечет за собой соответствующее изменение требований. Это особенно справедливо в случае разработки приложений для мобильных устройств, когда для понимания того, как люди будут использовать их в реальной жизни, могут потребоваться испытания в реальных условиях. Важно, чтобы проект предусматривал определенный уровень гибкости, позволяющий адаптироваться под обнаруживаемые в процессе разработки новые факторы и возникающие проблемы. Спецификация должна существовать, однако вначале она должна быть гибкой, допуская постепенное ее уточнение по мере обретения продуктом своей окончательной формы.

Очень важно выбрать тот реалистичный уровень организации процесса, которого вы и ваша группа сможете придерживаться. Даже если вы выполняете работу в индивидуальном порядке, желательно приучить себя к определенной самодисциплине. При работе над относительно небольшими проектами, каковыми обычно являются приложения для мобильных устройств, должен реализовываться подход, который будет привлекателен и удобен для всех участников группы. Чрезмерно детальное планирование лишает вас гибкости в тех случаях, когда первоначальные планы приходится изменять. Излишне строгие пункты плана или негибкие спецификации будут просто игнорироваться. С другой стороны, отсутствие плана или слабая организация процесса проектирования приведут к неустойчивости определения продукта, который либо опустится в своем качестве до посредственного уровня, либо должен будет переделываться. Существует "золотая середина", зависящая от размера, сферы применения и степени завершения разрабатываемого продукта. Важно правильно оценить масштабы проекта и, исходя из этого, наметить реалистичную организацию работы, позволяющую достигнуть конечной цели.

Существуют замечательные книги, посвященные вопросам организации процесса проектирования программного обеспечения, поэтому ниже мы ограничимся лишь краткими замечаниями. От себя лично могу дать следующие рекомендации, соблюдение которых, по моему мнению, играет важную роль при разработке программного обеспечения:

- *Вооружитесь определенной методологией разработки.* Методология — это план того, как вы будете проектировать и писать код. Оставшаяся часть настоящей главы посвящена разработке методологии для мобильных приложений.

- *Подготовьте план проекта в явном виде, оформив его в форме одного основного документа.* Любой проект только выиграет, если у вас будет один основной документ, с требованиями которого согласны все участники проекта. Шаблон такого плана проекта определяется ниже.
- *Планируйте периодические пересмотры проекта.* Всегда возникают непредвиденные проблемы и требуются изменения плана. Мобильные устройства не являются замкнутыми системами, и их успешное взаимодействие с пользователем и внешним миром в целом представляет собой сложный процесс; поэтому по мере продвижения работы вы будете все время узнавать что-то новое для себя. Лучше всего, если вы будете осознавать необходимость этого с самого начала и внесете в свой план соответствующие пункты.

Описание проекта

Для всех проектов разработки программного обеспечения, кроме самых тривиальных, важно иметь единственный "руководящий документ", в котором определяются: 1) требования проекта и целевое назначение завершенного продукта; 2) философия проекта; 3) архитектура приложения; 4) текущее состояние работ в данном направлении; 5) план, в соответствии с которым продукт будет переводиться из его текущего состояния в состояние успешного завершения. Для крупных проектов могут предусматриваться дополнительные документы, но всегда должен существовать один основной документ самого верхнего уровня, в котором определяются основные цели проекта, его нынешнее состояние и план работ. Этот документ должен быть реально действующим документом, который, по мнению всех участников группы, правильно определяет направления работы.

Единственный руководящий документ, который управляет всем процессом разработки, должен включать несколько разделов:

- *Цели проекта.* В этом разделе основного документа должно быть ясно указано, для чего предназначается завершенный продукт, и какая философия лежит в основе проекта.
- *Состояние проекта.* Этот раздел основного документа содержит точное описание текущего состояния проекта. Он является частью календарного плана проекта и позволяет судить о степени приближения работ над проектом к завершению. Здесь должны быть определены контрольные точки проекта, на достижение которых должны концентрироваться усилия, а также указано, какие элементы должны быть доработаны для завершения текущего контрольной точки процесса разработки. Здесь же должен предоставляться список наиболее значимых факторов риска или факторов, которые необходимо исследовать для разрешения соответствующих проблем.
- *Архитектура приложения и соответствующие диаграммы состояний.* Эта информация отражает технические аспекты плана. Для мобильных приложений этот раздел должен содержать соответствующие диаграммы состояний, описывающие дискретные состояния, в которых может находиться приложение, и связь этих состояний с объемами памяти и ресурсов, которые будут в ней храниться. (Далее об этом будет говориться более подробно.) Такой раздел фактически играет роль соглашения между всеми участниками группы разработчиков, в ко-

тором они обязуются придерживаться в своих реализациях установленных в нем требований. Если вы выступаете в роли единственного разработчика, то этот документ даст вам возможность оставаться честным перед самим собой; у каждого, кому довелось хотя бы однажды самостоятельно разрабатывать крупный проект, наверняка иногда возникало желание срезать тот или иной угол, чтобы добиться работоспособности средства, пусть даже это и будет в ущерб разумным принципам проектирования. Срезать углы гораздо сложнее, если перед вашими глазами находится соглашение, в котором указано, что вы должны в явной виде формулировать все свои предложения по ускорению работы над проектом. Этот раздел не должен быть чрезмерно длинным или сложным, ибо в противном случае выполнять его требования будет трудно, и им будут просто пренебрегать. В нем должно быть сформулировано, *что* необходимо сделать для того, чтобы проект удерживался в организационном русле, и, что еще важнее, в нем должны оперативно учитываться любые согласованные изменения проекта.

- *План разработки с указанием отдельных контрольных точек.* Еще более важную, чем календарный график, роль играет взвешенный план, устанавливающий дискретный набор контрольных точек проекта, которые должны проходиться по мере приближения проекта к завершению. По самому своему определению контрольные точки позволяют оценивать прогресс, достигаемый на пути к определенной конечной цели. Каждая контрольная точка представляет собой точку покоя, в которой вы можете остановиться, чтобы оценить состояние работ, подчистить код, который не был своевременно приведен в порядок, и при необходимости скорректировать проект. По мере выполнения проектных работ цели могут меняться, что влечет за собой необходимость корректировки контрольных точек; это допускается. В соответствии с эволюцией проекта может меняться его архитектура, что, в свою очередь, может потребовать внесения изменений в модель состояний; и это нормально. Можно почти не сомневаться, что пользовательский интерфейс также будет несколько раз переделываться. Отслеживание выполнения (или констатация невозможности выполнения) предварительно определенных контрольных точек является самым надежным мерилом прогресса, достигнутого в работе над проектом, и подсказывает, какие коррективы должны быть внесены по мере приближения к финишной линии. Контрольные точки проекта — ваши лучшие друзья.

Плановые пересмотры проекта

По мере продвижения работы над проектом вы будете открывать для себя много нового. Как бы вы ни старались все предугадать, всегда будут возникать непредвиденные трудности. Вы столкнетесь с проблемами производительности и вам, по всей видимости, не удастся сразу же получить наиболее подходящий вариант пользовательского интерфейса. Приобретение опыта в разработке мобильных приложений означает, что в будущем вы будете совершать меньше ошибок, но можно почти с полной уверенностью утверждать, что вообще избежать ошибок вам не удастся. Планируйте критический анализ полученных результатов после прохождения отдельных контрольных точек и особенно — завершающих этапов, когда вы можете подвести некоторые промежуточные итоги и идентифицировать основные проблемы, которые к этому времени могли проявиться. Итеративная природа процесса проектирования со-

временного программного обеспечения повышает значимость своевременного обновления основного документа проекта, а также методологии, способной предоставить эффективные возможности итерирования проекта.

Детали ничего не стоят, если общая картина неверна

Несмотря на то что это утверждение кажется очевидным, при написании кода мобильных приложений о нем часто забывают. Разработчики теряются в коде и забывают о том, для чего предназначено само приложение. В случае приложений для настольных компьютеров и Web-приложений это чаще всего проявляется в неудобстве и непонятности пользовательских интерфейсов, а также в отсутствии ключевых функций. Хотя приложение и работает, но не совсем так, как хотелось бы, и при его освоении конечного пользователя ожидают тяжелые испытания. Забывая о том, что именно должно собой представлять приложение в целом, вы рискуете утратить ясность целей.

Поскольку модели использования мобильных приложений фокусируются на выполнении лишь строго определенного круга задач в большей степени, чем это наблюдается для настольных приложений, результаты утери ясности целей разработчиком в этом случае оказываются еще более плачевными. Работая с неудачно спроектированным мобильным приложением, конечный пользователь в гораздо большей степени ощущает любые неудобства в работе и болезненно воспринимает любые недостатки приложения. В подобных случаях навигация в пределах приложения может превратиться в сущий кошмар, очень быстро обостряются проблемы производительности, и в результате всего этого конечный пользователь не испытывает ничего кроме разочарования. Поскольку размеры мобильных устройств близки к размерам обычных устройств и приспособлений, которыми можно оперировать одной рукой (часы, телефоны, складные ножи, музыкальные плееры, отвертки и тому подобное), пользователь невольно ожидает, что и их поведение должно быть столь же кристально прозрачным и интуитивно понятным. Мобильные приложения должны быть такими, чтобы пользователь мог их применять, не нуждаясь в особых инструкциях. Наверное, каждому из нас знакомы случаи, когда приходится пользоваться электронными часами или пультом дистанционного управления с многочисленными или неудачно расположенными кнопками. От того, насколько отчетливо разработчик представляет свои цели, зависит очень многое.

По этой причине очень важно, чтобы в основной документ вашего проекта были включены отчетливые формулировки целей создания приложения. Со временем эти цели могут претерпевать изменения, но действия всех участников проекта должны всегда оставаться согласованными. Если фокус проекта необходимо сместить в ту или иную сторону, то соответствующие изменения в свою работу должны внести все участники группы, и тогда на помощь приходит основной документ проекта с его разделом, содержащим четкие формулировки целей. Если ваше приложение не в состоянии обеспечить достижение поставленных целей, то, какой бы объем кода к этому времени ни был уже написан, успеха вы не добились.

Решайте задачи в определенной очередности; не бойтесь при необходимости возвращаться назад

Все является важным, но некоторые вещи важны более чем другие. В математическом анализе существует понятие эффектов первого, второго, ..., *n*-го порядка. Чем ниже порядок эффекта, тем большее влияние оказывает данный член ряда на поведение всей системы; эффектами высших порядков часто можно пренебречь. Аналогичная ситуация складывается и при разработке мобильных приложений; вы не имеете возможности уделять равное внимание всем аспектам, поэтому научитесь концентрировать свои усилия на том, что имеет наибольшее значение.

Ниже перечисляются шаги, описания которых демонстрируют, как подступиться к наиболее значимым аспектам вашего проекта мобильного приложения. Шаги представлены в порядке их следования. По мере внесения изменений в проект и при его переделках обязательно пересматривайте предыдущие шаги, ибо это позволит вам сохранять уверенность в том, что изменения, которые вы вносите для решения тех или иных проблем, не заставят вас переделывать ранее выполненную работу. Например, может сложиться такая ситуация, что решение проблем связи повлияет на способ взаимодействия пользователей с данными, предоставляемыми пользовательским интерфейсом; иногда это допустимо, иногда — нет. Если вы изменяете коммуникационную модель своего мобильного приложения, обязательно исследуйте, как это скажется на фундаментальных аспектах, управляющих пользовательским восприятием вашего приложения.

Шаг 0: прежде чем приступать к работе, определите сферу применения вашего приложения

Этот шаг назван нулевым, поскольку он действительно предшествует любой работе, связанной с проектированием и разработкой приложения. Прежде чем приступать к составлению планов и созданию программного обеспечения, вы должны хорошо представлять себе, каковым должен быть результат.

Приложения для настольных компьютеров подобны широкоугольным объективам в том смысле, что в типичных случаях они отображают значительный объем информации, который позволяют предоставлять пользователю экраны большого размера. В отличие от этого мобильные приложения напоминают увеличительное стекло или объектив с переменным фокусным расстоянием. Они предоставляют пользователю возможность быстро просматривать необходимые подробные данные, быстро переходить к ограниченным наборам данных и получать к ним доступ, а также принимать решения в реальном масштабе времени. Как правило, мобильные приложения предоставляют более специализированный набор сценариев по сравнению с приложениями, ориентированными на настольные компьютеры. Очень важно точно определиться с тем, на каких сценариях должно специализироваться ваше приложение.

Прежде чем приступать к реальной разработке приложения, определите подмножество функциональных средств, к которым пользователь сможет получать быстрый доступ в манере, свойственной мобильным устройствам. В случае создания нового приложения, аналога которого для настольных компьютеров не существует, выпишите ключевые сценарии, которые пользователи смогут выполнять с помощью вашего приложения, а также порядок действий пользователя, обеспечивающий использование этих сценариев на мобильном устройстве. Во многих случаях вам будет легче придать этим сценариям реальные очертания, если вы подготовите соответствующие рисунки или создадите прототипы. Если подразумевается определенная группа конечных пользователей, пообщайтесь с ними и предоставьте им возможность поработать некоторое время с экспериментальными версиями своих приложений, чтобы они могли дать о них свои отзывы.

Оптимальный подбор предоставляемых средств определяет все остальное

Если вы правильно выделите ключевые сценарии и возможности вашего приложения, то это окажет определяющее влияние на всю оставшуюся часть процесса разработки. Наличие явно сформулированного описания того, как конечные пользователи будут использовать ваше приложение, и детальное понимание их потребностей окажут вам неоценимую помощь при настройке производительности приложения, а также проектировании пользовательского интерфейса, коммуникационной системы и модели памяти.

Если вы не определите важнейшие с вашей точки зрения сценарии и возможности, то в результате вы получите бессистемную смесь средств, объединенных в одно приложение. Отсутствие явного списка основных функций приложения или разделения функций на группы в соответствии с их приоритетами приведет к тому, что пользовательский интерфейс не будет оптимизирован для эффективного решения ключевых задач. Например, если ожидается, что пользователь в основном будет заинтересован во вводе данных, то вы должны оптимизировать пользовательский интерфейс таким образом, чтобы обеспечить как можно более точное и надежное выполнение операций ввода. И наоборот, если ввод данных используется лишь изредка, то вариант пользовательского интерфейса ввода, оптимизированного не самым идеальным образом, может оказаться вполне допустимым, что позволит перебросить ресурсы проектирования и разработки на другие направления. Лишь только если группой разработчиков будут идентифицированы, перечислены и согласованы наиболее важные сценарии, эксплуатационные характеристики приложения могут быть настроены для их выполнения должным образом, а конечные пользователи не будут лишены важных для них средств из-за недосмотра.

Чтобы процесс разработки мог быть успешно завершен, составьте список ключевых требований, которым должно удовлетворять приложение, и возможностей, которые оно должно обеспечивать, и пусть этот список будет первым разделом вашего основного документа проекта.

Примеры удачных и неудачных описаний сценариев

Неудачное описание	*Удачное описание*
Приложение для обслуживания банковских операций "Обеспечить для пользователей мобильных устройств доступность Web-функциональности приложения MyBankFoos, предназначенного для мобильных банковских услуг, как в автономном, так и в сетевом режимах работы." *В этом описании ни слова не сказано о том, какие функциональные возможности являются наиболее важными. Возможность проверки состояния нужного счета? Оплата счетов? Хронология операций по переводу денежных средств? Смена обслуживающих банков? Денежные переводы? Операции в местах продажи? Заемные и залоговые условия при покупке автомобиля? Каковы те основные операции, в выполнении которых при помощи мобильного устройства больше всего будет нуждаться пользователь?*	**Приложение для обслуживания банковских операций** "1. Пользователи должны иметь возможность получать доступ к своим банковским счетам посредством не более пяти нажатий клавиш мобильного телефона, выполняемых одной рукой. 2. Пользователи должны иметь возможность совершать покупки и получать соответствующие подтверждения от торговых автоматов, используя инфракрасный порт устройства, с помощью не более трех клавиатурных нажатий." *Мы идентифицировали два ключевых сценария, которые хотели бы сделать доступными для пользователей.*
Опросы общественного мнения "Обеспечить замену бумаге и дощечке с зажимом при проведении опросов общественного мнения и избавиться от занесения данных опроса вручную." *Какие вопросы будут задаваться? Когда будет выполняться синхронизация данных?*	**Опросы общественного мнения** "Приложение должно предоставить пользователям возможность сбора информации в ходе опросов общественного мнения с помощью устройства Pocket PC. Вопросы будут предусматривать либо выбор одного из готовых вариантов ответа, либо простой цифровой ввод, а ответы будут кэшироваться на устройстве и отправляться на сервер после помещения устройства в лоток ПК. Опрос может содержать до 20 вопросов, а результаты, содержащие вплоть до 1000 ответов, должны храниться на устройстве. Ввод текста вручную не требуется." *Мы указали, какие виды вопросов должно обрабатывать приложение, а также каким образом будет осуществляться синхронизация устройства. (Здесь следует обратить внимание на то, что при составлении списка сценариев мы не заботились о том, какой именно способ сохранения результатов на устройстве будет использоваться или каков будет конкретный механизм синхронизации для работающего сценария – это важно только для нашей реализации, но не для конечного пользователя.) Мы указали также, чего не требуем от приложения (например, от него не требуется обработка свободного текста).*

Неудачное описание	*Удачное описание*
Инвентарный учет	**Инвентарный учет**
"Версия системы учета товара, ориентированная на настольные компьютеры, будет сделана доступной для подключенных к сети и автономных мобильных устройств." *Простой перенос функциональности Web-приложений и приложений, рассчитанных на настольные компьютеры, почти никогда не приводит к удовлетворительным результатам.*	"Приложение предназначено для использования на складах в режиме периодического доступа в сеть WI-Fi. Должна быть обеспечена возможность автономного режима работы с хранящимися на устройстве данными учета товаров, охватывающими вплоть до 5000 наименований. Идентификаторы товарных единиц должны сканироваться с помощью устройств для считывания штрих-кодов. Учетные записи о товарных единицах могут включаться в инвентарный список и исключаться из него. Устройства синхронизируются с использованием сети Wi-Fi, когда этого пожелает пользователь. Для обновления информации может быть затребован активный доступ к серверной базе данных. Ключевой сценарий: произвести сканирование при помощи устройства для считывания штрих-кодов и указать нажатием клавиши вид операции с инвентарным списком — добавление или исключение учетной записи. Произвести считывание порядкового номера покупки для его связывания с учетной записью о товарной единице. Ключевое требование: в случае невозможности считывания штрих-кода пользователь должен иметь возможность быстро ввести необходимую цифровую информацию вручную с помощью сенсорного экрана и пера. Для повышения надежности учета информация о неудачных попытках считывания штрих-кодов в процессе эксплуатации приложения должна заноситься в журнал." *Мы указали, в чем состоит суть ключевых требований, а также выписали основной сценарий, в соответствии с которым будет эксплуатироваться приложение.*
Игровые/обучающие приложения	**Игровые/обучающие приложения**
"Создать мобильное приложение для изучения слов иностранного языка." *Какая емкость словаря потребуется? Каким образом будет осуществляться процесс обучения в целом?*	"Приложение является игрой, предназначенной для проверки того, насколько хорошо пользователь знает слова иностранного языка.

Неудачное описание	*Удачное описание*
	Пользователям предлагаются вопросы, предлагающие выбрать из набора предложенных вариантов правильный перевод слова путем касания экрана. В устройстве может храниться до 1000 различных иностранных слов. В устройстве также должны храниться примеры предложений с изучаемыми словами." *Обучающая программа описана довольно полно. Потребуется дальнейшее исследование того, как должна работать игра, но мы уже определили высокоуровневую модель ввода-вывода и емкость словаря.*
Приложение для заказа авиабилетов "Приложение должно обеспечить сохранение и обработку в мобильном телефоне всей пользовательской информации о рейсах и сделать эту информацию доступной для работы в автономном режиме." *В этом описании мало говорится о том, что будет делать приложение, и каким образом пользователь будет работать с ним.*	**Приложение для заказа авиабилетов** "Пользователь должен иметь возможность получения доступа к сохраняемой в устройстве информации, относящейся к электронному заказу билетов. Информация, которую сможет получать пользователь мобильного телефона, включает в себя номер заказа, номер рейса, время вылета и данные об аэропортах вылета и назначения, причем для получения доступа к этой информации должно требоваться не более трех клавиатурных нажатий. Должен быть обеспечен быстрый вызов и передача этой информации работнику, занимающемуся резервированием посадочных мест."

Шаг 1: начните с анализа проблем производительности и никогда не упускайте их из виду

Идентифицировав ключевые сценарии и возможности мобильного приложения, вы должны позаботиться о создании наилучших условий работы для тех, кто будет им пользоваться. Это означает, что в процессе проектирования и создания кода мобильного приложения вы должны оптимизировать его производительность и интерактивные свойства.

- Вы должны определить общие показатели, характеризующие способность вашего приложения реагировать на действия пользователя. Например: "На появление диалога не должно уходить более 0,5 секунды", "Визуально определяемая реакция на нажатие клавиши должна следовать немедленно" или "Раскрытие узлов дерева не должно занимать более 0,15 секунды". Здесь были приведены в качестве примера неплохие значения показателей, к достижению которых следует стремиться.

- Вы должны определить конкретные числовые показатели для ключевых сценариев. Например: “Загрузка данных инвентаризационной ведомости ни при каких обстоятельствах не должна останавливать работу пользовательского интерфейса на более чем 0,2 секунды без появления курсора ожидания” или “Длительность загрузки данных инвентаризационной ведомости ни в коем случае не должна превышать 3 секунд”.

Тестируйте свои допущения в реальных условиях эксплуатации сценариев. Даже если в вашем распоряжении еще не имеются все необходимые реальные данные или код, легко тестировать эффекты задержек при выполнении пользователем часто встречающихся задач, имитируя их путем вставки состояний ожидания в скелет кода пользовательского интерфейса приложения. Очень важно хорошо себе представлять, какие периоды ожидания приемлемы, а какие — нет. Следует стремиться к тому, чтобы приложение всегда предоставляло немедленное подтверждение того, что та или иная операция находится в состоянии выполнения, если она не может быть завершена очень быстро. Кроме того, если работа должна выполняться в течение длительного или неопределенного времени, ее выполнение может быть поручено фоновому потоку.

Разрабатывая приложение, используйте реалистические размеры данных, которые либо соответствуют предполагаемым размерам данных, с которыми придется работать пользователю, либо превышают их. Одной из распространенных ошибок, допускаемых разработчиками в процессе создания приложений, является использование тестовых данных небольшого объема, и иногда это приводит к тому, что при работе с реальными объемами фактических данных приложение функционирует из рук вон плохо. Чем раньше вы начнете тестировать приложение на реальных данных, тем раньше эти проблемы всплывут на поверхность и тем выше вероятность того, что вам удастся справиться с проблемами производительности.

Блок-схема организации процесса разработки мобильного приложения представлена на рис. 4.1. Если вы столкнулись с проблемами производительности — *остановитесь*! Можно сформулировать это и по-другому: если в процессе проектирования и разработки приложения вы столкнулись с проблемами производительности — *немедленно прекратите дальнейшее написание кода*! Слишком уж часто разработчики с головой погружаются в это занятие, стремясь поскорее получить “завершенный код” и давая себе слово, что к решению возникших проблем производительности они после этого обязательно вернутся. В лучшем случае такая стратегия является рискованной! Она редко приводит к успеху при разработке приложений для настольных компьютеров и серверов и обычно дает довольно-таки плачевные результаты; в случае же мобильных устройств результаты ее применения будут еще худшими. Причина этого заключается в том, что проблемы производительности часто коренятся не в недостатках какого-то отдельного алгоритма, который можно просто доработать или оптимизировать, не влияя на остальные части системы. До тех пор пока вы детально не выясните природу проблемы и не убедитесь в том, что альтернативное решение способно ее устранить, вы не можете сказать фактически ничего определенного о том объеме переработки проекта, который для этого может потребоваться. Вам может повезти, однако везение довольно быстро покидает тех программистов, которые слишком сильно полагаются на это.

Рис. 4.1. *Методология, подчиненная требованиям производительности*

Во многих случаях проблемы производительности носят систематический характер и касаются всего приложения в целом. Систематическое снижение производительности может быть, в частности, обусловлено способом обмена данными с памятью, количество одновременно удерживаемых в памяти ресурсов, а также способом перерисовки и обновления информации на экране, который используется пользовательским интерфейсом. Проблемы производительности подобного рода обычно появляются из-за неудачного выбора основных проектных решений, и, чтобы избежать их, необходимо позаботиться об этом уже на самых ранних стадиях процесса проектирования и разработки приложения. Потери от чрезмерно быстрого продвижения вперед для получения "завершенного кода" и последующего возврата к пересмотру стратегии доступа к данным, повторному проектированию модели использования памяти и организации выполнения определенной работы в фоновом режиме для улучшения интерактивности пользовательского интерфейса могут быть поистине огромными. Даже если в результате внесения запоздалых проектных изменений вы и получите работающий код, он будет плохо организован. Реальные основы высокой производительности приложения должны закладываться еще в его фундаменте.

Не забывайте также о том, что по мере разработки приложение будет постепенно разрастаться и усложняться. Вы буде писать все больше кода, создавать все новые объ-

екты, и у вас будет появляться все больше компонентов, конкурирующих между собой за право обладания ограниченным пулом ресурсов. Если вы столкнетесь с проблемами производительности приложения и не займетесь ими сразу же, как только они замечены, то по мере того, как объем кода будет расти, эти проблемы будут только усугубляться, а не ослабляться. К тому же добавление нового кода будет порождать дополнительные зависимости в моделях данных, памяти, пользовательского интерфейса, коммуникационной модели и других логических частях программы, которые вы написали. Откладывая решение проблем "на потом", вы сами себя загоняете в угол.

Нередко разработчики рассуждают таким, на первый взгляд вполне разумным, но на самом деле глубоко ошибочным образом: "Понять, какие части приложения нуждаются в улучшении, можно лишь только после того, как будет написан весь код". Такая позиция весьма порочна, поскольку в ней содержится предположение о том, что отдельные части вашего приложения каким-то удивительным образом не зависят друг от друга. Вместе с тем, когда вы напишете весь код, в него будет введено множество явных и неявных зависимостей между различными системами, и вы, вероятнее всего, будете в состоянии внести в код лишь некоторые изменения количественного, но не качественного характера. Вам не уйти от этого даже в том случае, если вы приложите максимум усилий к инкапсуляции кода приложения. Чем больше объем кода, тем больше в нем существует зависимостей между отдельными частями; пытайтесь бороться с этим, тщательно продумывая структуру приложения, но знайте, что таковы реалии жизни. Браться за решение проблем производительности следует тогда, когда базовый код еще обладает достаточной гибкостью и остается открытым для реструктуризации. Лучше всего заниматься этим сразу же по ходу написания кода.

Как только у вас возникнут проблемы с производительностью — остановитесь, оцените ситуацию и выясните, что именно происходит. Не связано ли это с неэффективным обновлением пользовательского интерфейса? Не хранится ли в памяти слишком много данных, в результате чего "сборщику мусора" приходится непрестанно трудиться? Не является ли данный вид обработки длительным изначально длительным в силу своих особенностей, в результате чего такую обработку целесообразнее выполнять в фоновом режиме, асинхронно по отношению к пользовательскому интерфейсу? Не работаете ли вы с большими объемами данных, используя высокоуровневую объектную модель с сохранением состояний, тогда как лучше было бы использовать низкоуровневую программную модель без сохранения состояний? Определите, в чем коренятся причины проблемы, и соответствующим образом перестройте структуру кода для их устранения, прежде чем далее создавать код, который будет зависеть от принятых вами допущений.

Придерживаясь в процессе разработки приложения определенной дисциплины и не забывая о производительности, вы сможете добиться того, что код, а также модели использования данных и памяти вашего приложения не будут содержать ничего лишнего, а дизайн пользовательского интерфейса будет отличаться ясностью и эффективностью.

Высокая производительность — основная предпосылка создания удобных условий работы для пользователя

(если производительность низка, выполнение последующих шагов вам ничего не даст)

Если вы создадите привлекательный и интуитивно понятный интерфейс, простую и надежную коммуникационную модель и объектно-ориентированную модель данных, но ваше приложение будет вести себя неудачным с точки зрения конечных пользователем образом, то они будут разочарованы. В силу самой природы мобильных устройств конечные пользователи рассчитывают на их высокую интерактивность. Люди всегда носят эти устройства при себе и надеются, что смогут воспользоваться ими сразу же, как только в этом возникнет необходимость.

Очень важно понимать, что восприятие любых эксплуатационных параметров приложения носит исключительно субъективный характер. Пользователя не интересует, сколько времени ушло на получение данных, ему важно лишь то, насколько длительным показался ему этот промежуток, и что в это время происходило. Это обстоятельство может быть выгодно использовано при проектировании приложения. Длительно выполняющиеся операции не будут казаться большим злом, если поведение приложения таково, что у пользователей складывается впечатление, будто они сохраняют контроль над ситуацией и ощущают ответную реакцию приложения. Главные доказательства высокой производительности приложения заключены в том, каким его воспринимает реальный пользователя.

Поставьте перед собой формирование нужного вам пользовательского восприятия производительности приложения в качестве основной цели. Эту задачу вы должны начать решать еще до того, как приступите к проектированию пользовательского интерфейса, многие аспекты которого будут определяться требованиями производительности. Например, предположим, что в приложении предусмотрена запрашиваемая пользователем операция, для выполнения которой требуется 20 секунд. Поскольку оставить пользовательский интерфейс без возможности реагировать на действия пользователя на целых 20 секунд было бы недопустимо, эта работа должна выполняться в фоновом режиме, чтобы в процессе обработки запроса пользовательский интерфейс сохранял свою интерактивность. Если вы поступите именно так, то часть визуального пространства, занимаемого пользовательским интерфейсом, целесообразно выделить для вывода информации о состоянии выполнения данной операции, чтобы пользователь знал, что его запрос принят и обрабатывается.

Пример разработки пользовательского интерфейса с учетом обеспечения высокой производительности

Как ActiveSync, так и Internet Explorer для Pocket PC и интеллектуальных телефонов предлагают элементы пользовательского интерфейса, назначение которых состоит в том, чтобы информировать пользователей о состоянии выполнения инициированных ими асинхронных запросов. Благодаря этому пользовательский интерфейс сохраняет свою интерактивность, и у пользователя не возникает чувства потери контакта с приложением.

Для синхронизации календарной информации с сервером Exchange Server приложению ActiveSync может требоваться телефонный звонок или иная форма операции соединения, на выполнение которой уходит длительное время. В процессе выполнения такой операции пользовательский интерфейс устройства отображает пояснительный текст,

информирующий пользователя о текущем состоянии попытки создания соединения. Таким текстом может быть “Набор номера …”, “Соединение с сервером”, “Синхронизация расписания”, “Загружено 12 из 20” и тому подобное. Такие несложные уведомляющие элементы пользовательского интерфейса не дают никакого выигрыша в производительности, но зато сохраняют в пользователе уверенность в том, что полезная для него работа продолжает выполняться, и удерживают его от попыток отмены запроса из-за того, что ответ на него затягивается.

Браузер Pocket Internet Explorer также может нуждаться в соединении с источниками данных, поиске IP-адресов URL и выполнении других задач в процессе загрузки Web-страниц. При выполнении подобных затяжных задач заголовок окна браузера обновляется, постоянно отображая текст, информирующий пользователя о состоянии соединения. Кроме того, при загрузке Web-содержимого отображается анимация флага Windows. Оба указанных действия информируют пользователя о том, что выполнение операции продолжается.

В обоих случаях пользовательский интерфейс сохраняет свою интерактивность, а пользователям предоставляется возможность отменить операцию, если они того захотят. Наличие возможности в любой момент прекратить выполнение операции поддерживает в пользователе чувство уверенности в сохранении контроля над ситуацией, а непрерывное получение им информации о состоянии выполнения операции уменьшает вероятность того, что он ее отменит. Конечно, было бы лучше, если бы такие операции выполнялись мгновенно, но в тех случаях, когда это невозможно, следующее наилучшее решение заключается в поддержании интерактивности пользовательского интерфейса и непрерывном информировании пользователя о развитии ситуации.

Шаг 2: спроектируйте подходящий пользовательский интерфейс

Как ранее уже отмечалось, прежде чем приступить к проектированию мобильного приложения, важно выделить ключевые сценарии и возможности, которые будут определять сферу его применения. Далее, процесс разработки должен быть ориентирован на достижение высокой производительности приложения, ибо это будет гарантией того, что при составлении кода вы не загоните сами себя в угол. Если вы не в состоянии добиться того, чтобы выбранные вами ключевые сценарии и средства функционировали удовлетворительно, пересмотрите проект.

Следующей по очередности в списке важных задач стоит разработка подходящего пользовательского интерфейса. Подобно тому, как проблемы производительности решаются с учетом особенностей человеческого восприятия, отлично функционирующий пользовательский интерфейс должен проектироваться с учетом специфики целевых устройств. Пользовательский интерфейс, являющийся оптимальным для одного класса устройств, для устройств другого класса может оказаться далеко не лучшим. Учет специфики устройства всегда себя оправдывает.

Будьте готовы к необходимости итеративного проектирования пользовательского интерфейса, особенно если вы только начинаете работать с мобильными устройств или приступаете к работе с новым классом устройств, форм-фактор которых отличен от тех, с которыми вы имели дело до этого. По мере накопления опыта работы с устройствами данного класса вы сможете проектировать пользовательские интерфейсы в более сжатые сроки, но вам все равно придется возвращаться к предыдущим этапам работы, чтобы все было сделано наилучшим образом.

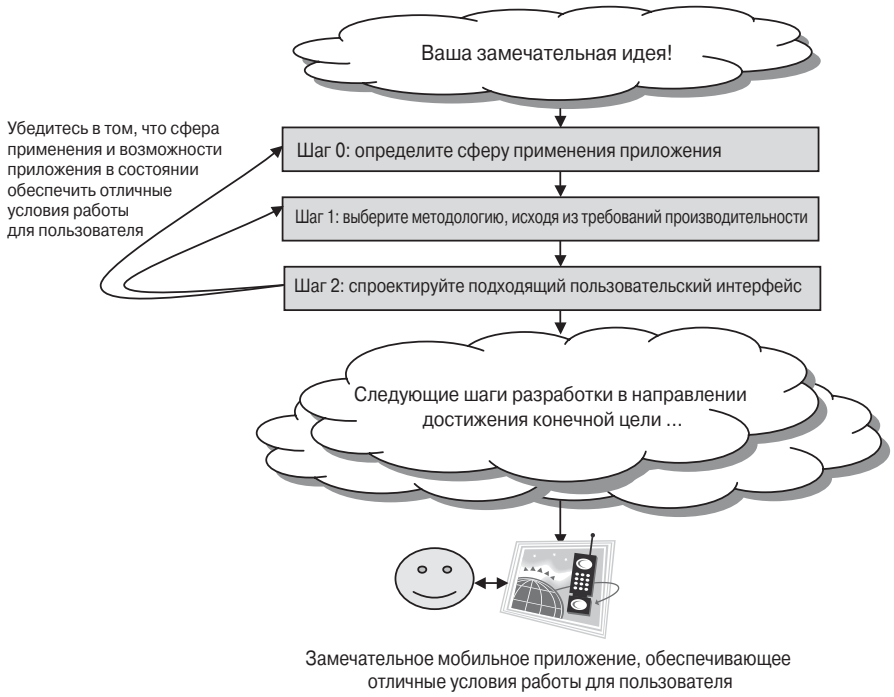

Рис. 4.2. *Проектирование пользовательского интерфейса, ориентированное на достижение высокой производительности*

Будет неплохо, если вы возьмете за правило отделять логику выполнения приложения от логики представления информации, в результате чего внесение изменений в пользовательский интерфейс не будет влечь за собой необходимости изменения логики приложения. Это вдвойне справедливо, когда речь идет об устройствах. Абстрагируя логику выполнения приложения от логики представления информации, вы усиливаете свои позиции как в отношении пересмотра и улучшения пользовательского интерфейса для конкретного целевого устройства, так и в отношении быстрого переноса приложения на новые классы устройств. Сегодняшнее приложение для устройств PDA завтра вполне может стать приложением для смартфонов, если только код его пользовательского интерфейса тесно не переплетен с основной логикой приложения. Тщательное разделение логики приложения и логики интерфейса принесет вам неплохие дивиденды как в плане сопровождения кода, так и в плане его переносимости.

Удачный пользовательский интерфейс — довольные пользователи (неудачный пользовательский интерфейс — ежедневный источник раздражения)

Ключевыми факторами проектирования пользовательского интерфейса, определяющими его успешность, являются обеспечение продуктивной работы конечного пользователя и сохранение постоянной способности интерфейса к интерактивному взаимодействию с пользователем. Очень важно, чтобы пользователи могли быстро выполнять основные сценарии работы приложения. Например, если конечным пользователям приходится часто вводить календарные даты, то используемые для этого

способы ввода данных должны обеспечивать как можно более высокую скорость, простоту и предсказуемость этого процесса. Если в других возможных ситуациях пользователям приходится часто выбирать элементы из длинного списка, то быстрее всего это можно сделать не тогда, когда все элементы отображаются в одном списке типа `ListBox`, а тогда, когда для этого разработан специальный графический пользовательский интерфейс, с помощью которого пользователи смогут быстро осуществить поиск нужного элемента. Простой перечень автомобильных запчастей только выиграет, если дополнить его схематическим изображением автомобиля, прикосновение к отдельным частям которого на дисплее будет перемещать вас в нужную часть списка. Аналогичным образом в медицинском приложении может быть использовано схематическое изображение человеческого тела. Корректный пользовательский интерфейс зависит от типа решаемых задач и специфики устройств, на которых выполняется приложение. Вот почему реализовать идею пользовательских интерфейсов, которые "пишутся однажды, выполняются везде", в случае мобильных устройств не всегда удается. Подходы такого типа просто не в состоянии обеспечить одинаково хорошие условия работы для пользователей устройств с различными размерами дисплеев и возможностями ввода.

С обеспечением высокой продуктивности работы пользователя тесно связано поддержание способности интерфейса к интерактивному взаимодействию. Пользовательский интерфейс приложений для мобильных устройств должен характеризоваться быстрым откликом. Это вовсе не означает, что пользователя вообще нельзя оставлять в состоянии ожидания; избежать этого иногда просто невозможно. Однако ни в коем случае нельзя заставлять пользователя лишь догадываться о том, выполняется ли запрошенная им операция или запрос необходимо повторить. Отсутствие каких-либо признаков активности устройства, получившего запрос на выполнение операции, вызывает у пользователей раздражение, поскольку психологически они настроены на то, что после нажатия кнопки, касания экрана или иного воздействия на органы управления находящегося у них в руках устройстве, должно обязательно что-то произойти.

Шаг 3: выберите подходящие модели данных и памяти

Выбранные вами для мобильного приложения модели данных и памяти определяют, каким образом будет осуществляться управление объектами и ресурсами, хранящимися в памяти. И наоборот, модели памяти определяют, каким образом ваше приложение будет избавляться от ненужных данных и ресурсов, чтобы освободить память для своих нужд. Мобильные устройства заметно отличаются от своих настольных собратьев повышенными требованиями к эффективности управления данными и памятью.

В случае мобильных устройств крупными пулами памяти и файлами подкачки жертвуют ради уменьшения их размеров и снижения энергопотребления. Приложение, выполняющееся на мобильном устройстве, обитает уже не в роскошном особняке, а просто в хорошей городской квартире. Вместо спортивного автомобиля, используемого приложениями настольных компьютеров для быстрого объезда окрестностей, теперь имеется только мотоцикл. Считать, что ваше мобильное приложение "обитает" в условиях намного более ограниченного пространства — неплохая аналогия, которую полезно постоянно держать в голове в процессе проектирования моделей данных и памяти.

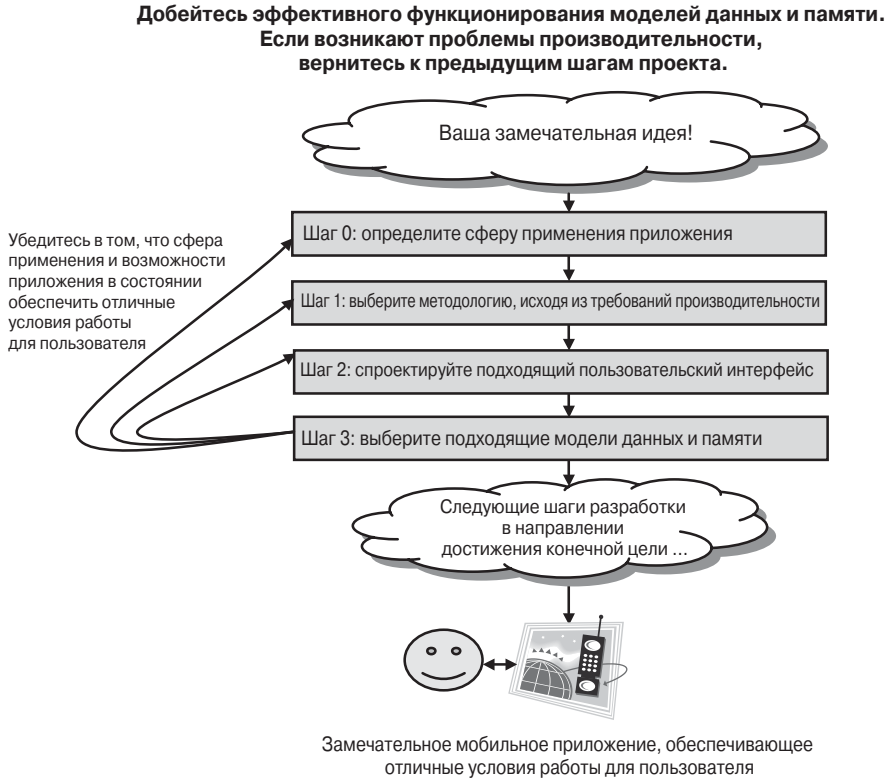

Рис. 4.3. *Проектирование моделей данных и памяти, ориентированное на достижение высокой производительности*

Те, кто программирует приложения для настольных компьютеров, исключая приложения, требующие огромных ресурсов памяти (например, сложные программы рисования часто обрабатывают числовые матрицы очень больших размерностей), в своем большинстве обычно даже не задумываются о том, какую модель памяти лучше использовать. Поэтому систематическое и заблаговременное освобождение памяти от хранящихся в ней ресурсов, необходимость в которых отпала, как правило, не производится. Любые необходимые данные сразу же загружаются в память без предварительной ее очистки от ненужных данных. Непрерывная загрузка данных и ресурсов в память продолжается либо из-за беспечности программиста, либо исходя из того, что они еще могут понадобиться пользователю. Если уж приложение позаботилось о загрузке некоторых данных или изображений из сетевого ресурса, то почему бы не подержать их в памяти подольше, чтобы сразу же предоставить их пользователю, если ему захочется вновь обратиться к этому же ресурсу? В случае приложений, предназначенных для настольных компьютеров, такой подход является в значительной степени оправданным; находящиеся в памяти неиспользуемые данные, в конечном счете, сбрасываются на жесткий диск, а вызов необходимой нужной страницы данных в память выполняется гораздо быстрее, чем повторное подключение к сети и посылка запроса. В распоряжении у приложения имеется целый дом, и неиспользуемые вещи можно просто-напросто разместить где-то на чердаке.

Как нами ранее уже обсуждалось, в случае мобильных устройств наблюдается совершенно иная ситуация. Учитывая ограниченность объема доступной памяти и отсутствие вспомогательных накопителей, которые можно было бы использовать для временного хранения страниц памяти, разработчики обязаны использовать память и ресурсы весьма расчетливо. Разработчик мобильного приложения обязан целенаправленно выбрать модель данных, в соответствии с которой будет осуществляться управление хранением данных в памяти и сборкой мусора. В случае устройств правильная стратегия часто заключается в освобождении памяти от данных или явном их перемещении на карту флэш-памяти и повторном вызове данных в память, когда они вновь потребуются.

Удачная модель данных означает высокую производительность и гибкость дизайна

(неудачная модель — заведомо низкую производительность)

Если согласиться с тем, что существует искусство создания замечательных приложений, то важнейшей его составляющей является умение выбирать наиболее подходящие модели данных. Рациональное управление памятью и ресурсами подразумевает следующее: 1) приобретение навыков экономного расходования памяти и хранения в ней только тех объектов, в которых действительно существует острая необходимость и которые будут немедленно использоваться, 2) использование кэширования важных данных на устройстве, но вне активной памяти и 3) перемещение остальных данных на накопитель, находящийся за пределами устройства. Научившись этим трем вещам, вы пройдете значительную часть дистанции, отделяющей вас от создания великолепно функционирующих приложений, работа с которыми будет доставлять удовольствие пользователям.

Шаг 4: выберите подходящую модель коммуникации и модель ввода–вывода

Выбор коммуникационной модели, а также модели ввода-вывода определяет, каким образом ваше приложение будет связываться с ресурсами, хранящимися вне текущего процесса. В качестве таковых могут выступать либо локальные ресурсы устройства, например, хранящиеся на нем файлы и базы данных, либо ресурсы, являющиеся внешними по отношению к физическому устройству, например такие, доступ к которым осуществляется посредством связи через сокеты, или же такие, как файлы на серверах, Web-службы и удаленные базы данных. Выбор способов, используемых приложением для связи как с локальными, так и с удаленными ресурсами оказывает большое влияние на восприятие приложения пользователями, и поэтому занимает одно из ведущих мест в списке всего того, что должно быть включено в вашу методологию разработки.

Работа с локальными ресурсами устройства

Почти все приложения, способные предоставлять пользователю информацию, хранящуюся в течение длительного времени, обеспечивают эту возможность путем локального сохранения и извлечения информации на устройстве.

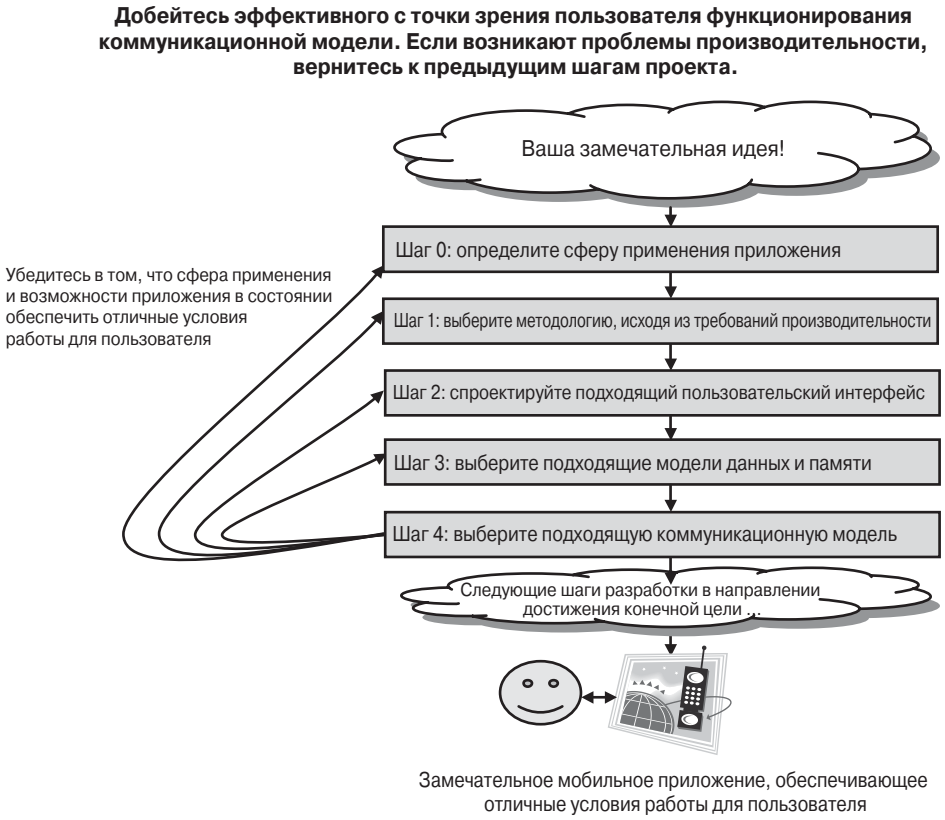

Рис. 4.4. *Проектирование коммуникационной модели, ориентированное на достижение высокой производительности*

Наиболее важными факторами, оказывающими влияние на работу с локальными данными устройства, являются формат данных и уровень абстракции программной модели, используемой для работы с этими данными.

Формат данных

Вообще говоря, формат, в котором хранятся данные, выбирается на основе компромисса между требованиями эффективности и удобства использования. Любые конкретные данные могут храниться в виде двоичных файлов, простых текстовых файлов, текстовых XML-файлов или в виде структурированных таблиц локальных баз данных. Требования эффективности означают минимизацию размера данных и обеспечение максимально возможной производительности приложения. К числу требований удобства использования относятся максимально возможное повышение производительности труда разработчика, улучшение условий сопровождения кода, минимизация объема необходимого тестирования, обеспечение возможности обмена данными между приложениями и гибкость формата, обеспечивающая возможность его последующего расширения.

Двоичные форматы хранения данных предлагают самые широкие возможности как в отношении снижения размера данных, так и в отношении повышения произво-

дительности приложения. По этой причине данные, характеризующиеся большой плотностью информации, например, изображения, чаще всего сохраняются в двоичных форматах. Потребности в сохранении данных изображения настолько специфичны, что для этого имеется целый ряд популярных форматов, каждый из которых предлагает свой вариант достижения компромисса между размером данных, производительностью и точностью передачи изображения. Каждый из двоичных форматов изображения отвечает определенным запросам. Двоичный формат может использоваться для хранения не только изображений, но и данных произвольной природы. Однако работать с двоичными данными труднее; если вы создаете собственные двоичные форматы, то у вас появятся заботы, связанные с необходимостью учета различий в версиях данных и обеспечением возможности использования этих данных другими приложениями.

Хранение данных в текстовых форматах значительно облегчает их использование и расширяет возможности их переноса в другие приложения, так как декодировать их легче. Однако размеры текстовых файлов больше по сравнению с их двоичными аналогами. Размеры XML-файлов оказываются еще бо́льшими, чем размеры обычных текстовых файлов, поскольку текстовые данные в них дополняются информацией о схеме данных. Эти дополнительные метаданные схемы значительно повышают гибкость данных в отношении учета их версий и переносимости в другие приложения, но требуют использования дополнительного пространства. Кроме того, при чтении и записи XML-файлов их необходимо дополнительно пропускать через синтаксические анализаторы, что усложняет их обработку по сравнению с обычными текстовыми файлами, в которых для разделения данных используются запятые или символы табуляции. Отмеченная гибкость достается за счет дополнительных накладных расходов. Эти дополнительные расходы можно снизить, используя разумные стратегии реализации, но полностью избавиться от них невозможно.

Базы данных предлагают наивысшую степень организации данных, однако привносят с собой дополнительные накладные расходы, связанные с выполнением процессора базы данных.

Различные уровни абстракции программной модели

Обычно программные модели, предназначенные для работы с сохраненными данными, имеют несколько уровней. Так, для работы с файлами в .NET Compact Framework предлагаются следующие уровни абстракции, перечисленные в порядке их повышения:

- Двоичные потоки.
- Текстовые потоки.
- Объекты однонаправленного чтения и записи XML.
- Объектная модель документов (DOM) XML.

Каждый из указанных уровней предлагает все более высокий уровень абстракции для облегчения работы с данными, что связано с соответствующим увеличением накладных расходов. В некоторых случаях эти накладные расходы пренебрежимо малы и вполне оправдывают то повышение производительности труда разработчика и степени надежности, которое обеспечивают протестированные высокоуровневые API-интерфейсы. В других случаях, особенно при работе с большими объемами данных,

высокоуровневые абстракции выдвигают такие дополнительные требования к памяти и процедуре разработке, которые являются неприемлемыми. В подобных случаях разработчикам следует переходить на один уровень абстракции ниже в стеке API-интерфейсов и попытаться решить возникшие проблемы с использованием API-интерфейса более низкого уровня, который характеризуется меньшими накладными расходами. Важно уметь оценивать, какие накладные расходы связаны с применением того или иного уровня абстракции.

Выбор формата хранения данных и программной модели

Какой формат данных следует использовать для хранения данных — целиком зависит от целей вашего приложения; никакого универсального рецепта здесь не существует. Распространенной ошибкой тех, кто только приступает к разработке программного обеспечения для мобильных устройств, является допущение о том, что, поскольку в этом случае приходится иметь дело с ограниченными ресурсами, следует сразу же переходить на самый низкий уровень абстракции и использовать двоичные файлы наряду с потоковыми операциями файлового ввода-вывода. Иногда необходимость в этом действительно существует, но в большинстве случаев это просто означает выполнение ненужной работы, которая потребует дополнительного тестирования и, вероятно, приведет к худшему решению, не обеспечивающему достаточной гибкости. Общее правило заключается в том, чтобы использовать наивысший уровень абстракции, допустимый с точки зрения размера данных и достигаемой при этом производительности. Было бы неразумно изобретать собственный двоичный формат для данных сравнительно небольшого объема, поскольку при средних запросах памяти лучшего варианта, чем XML, не найти. С XML легко работать, он обеспечивает надежную работу с различными версиями данных и для него существует высокоуровневый API-интерфейс, упрощающий процесс программирования. Точно так же, в случае возникновения действительной необходимости в двоичном формате, например, для хранения больших объемов данных, описывающих изображения, гораздо предпочтительнее воспользоваться уже имеющимися и проверенными на практике форматами, если таковые имеются. Поскольку существует целый ряд хорошо зарекомендовавших себя форматов изображения, изобретение собственного формата будет, как правило, напрасной тратой времени. При любой удобной возможности старайтесь использовать уже существующие компоненты и форматы данных; изобретайте свои собственные форматы лишь в тех случаях, когда вы убеждены, что высокоуровневые подходы не сработают.

Работа с внешними по отношению к устройству ресурсами

Не считая простейших игр и элементарных приложений вспомогательного характера наподобие калькуляторов, большинство представляющих интерес приложений для мобильных устройств, так или иначе, взаимодействуют с данными, хранящимися за пределами устройств. Эти данные могут находиться в базе данных, реплицируемой на устройстве. Они могут содержаться в хранящейся на устройстве адресной книге, синхронизируемой с электронным почтовым сервером. Ими также могут быть изображения или музыкальные файлы, загруженные с Web или настольного компьютера или “полученные” от другого устройства через инфракрасный порт. Пользователи мобильных телефонов обмениваются SMS-сообщениями. XML-данные передаются Web-службам и принимаются от них.

Доступ к данным может осуществляться через сеть Wi-Fi, данные могут передаваться посредством флэш-карты, вставленной в устройство, пересылаться при помощи мобильных телефонов с использованием протокола GPRS, передаваться через инфракрасный порт или просто пересылаться по Ethernet-кабелю, подключенному к устройству. Короче говоря, существует множество самых различных типов данных, которыми устройства могут обмениваться, и средств, обеспечивающих передачу этих данных.

Остановив свой выбор на нескольких сценариях, выполнение которых будет обеспечивать ваше приложение, проникнувшись решимостью не забывать в процессе разработки приложения о необходимости обеспечения высокой производительности, подготовив прототипы пользовательского интерфейса, который предоставит пользовать возможность взаимодействовать с приложением, вы должны уделить внимание выбору коммуникационной модели.

В процессе разработки коммуникационной модели вы должны обязательно обратиться к имеющимся основным сценариям вашего приложения и определить, какие именно соединения потребуются для активизации этих сценариев. К числу вопросов, на которые вы должны дать ответы, относятся следующие:

- Должно ли быть подключение к источникам актуальных данных постоянным или оно будет требоваться лишь время от времени?
- Какие объемы данных потребуется перемещать? От ответа на этот вопрос могут зависеть другие ваши решения.
- Каким образом будет осуществляться обмен данными? Посредством беспроводного сетевого соединения? Посредством лотка ПК? Посредством карты флэш-памяти?
- Какова структура расходов на обеспечение соединений? Поскольку за использование сети Wi-Fi в составе интрасети ничего платить не надо, плата за использование мобильных сетей обычно взимается в соответствии с объемом передаваемых данных.

Производительность и надежность

Вы должны исходить из того, что в мобильных сетях соединения часто разрываются, и, как правило, это происходит в самый неподходящий момент. Даже если устройство подключено к сети непосредственно через кабель, все равно следует подумать о мерах предосторожности, заранее предполагая, что случиться может все что угодно: сервер может выйти из строя, кабель может порваться, прокси-серверы, преобразователи сетевых адресов, маршрутизаторы и брандмауэры могут внезапно “организовать против вас заговор”, а некие противные маленькие злые гномики могут проникнуть в “железо” и атаковать ваши пакеты данных крохотными молоточками, чтобы не дать им дойти до пункта назначения. Разумная доля скептицизма, граничащая с паранойей, сторицей себя окупит. Заблаговременно предполагайте любые мыслимые и немыслимые ситуации, лишь бы ваше приложение было надежно защищено, и трезво подходите к реалиям работы в условиях сбойных соединений и возникновения непредвиденных затруднений.

Большинству из нас приходилось сталкиваться с тем, что выполнение приложения на сервере или настольном компьютере на некоторое время приостанавливается из-за недоступности сетевого ресурса. Обычно такое приложение либо зависает, либо не

подает признаков жизни в течение времени, которое тянется целую вечность, пока, наконец, управление не будет вновь возвращено пользователю. В случае мобильных устройств ситуация заведомо худшая. Поезда заходят в туннели, а двери лифтов захлопываются, в результате чего сигналы, которые до этого были доступными, блокируются, но никаких предупреждающих сообщений об этом выполняющееся мобильное приложение получить не может. Пропадает связь между отдельными ячейками сотовой связи, отключаются базовые станции сетей Wi-Fi, и вообще, иногда бывает так, что что-то просто идет наперекосяк по каким-то необъяснимым причинам.

Надеюсь, я был достаточно убедителен, чтобы вы прониклись убежденностью в необходимости написания соответствующих кодов, защищающих приложение от подобных неприятностей. Существуют способы, которые облегчают решение этой задачи:

- Выполняйте все сетевые операции в асинхронном режиме, что позволит сохранить постоянную готовность пользовательского интерфейса к отклику и предоставит пользователям возможность в необходимых случаях отменить запрос и заняться другой работой.
- Поместите все процедуры доступа к сети в блоки `try/catch` (то есть организуйте структурную обработку исключений). Исходите из того, что время от времени будет выполняться каждый из этих `catch`-блоков, и имитируйте эти ситуации в процессе проектирования и тестирования приложения, чтобы проверить, правильно ли реагируют на возникновение исключений предусмотренные для них обработчики. Целесообразно предусмотреть, чтобы исключения возбуждались при любых периодических нарушениях связи; точно так же, целесообразно перехватывать подобные исключения и обрабатывать их так, чтобы это облегчало работу пользователя.

Доступ к сетевым ресурсам намного расширяет возможности мобильных приложений, но при этом неизбежно привносит в ваше приложение элементы, контролировать которые вы не в состоянии. Очень важно, чтобы мобильное приложение могло надежно вести себя в тех весьма реальных ситуациях, когда попытка получения доступа к внешним ресурсам оказывается неудачной или может быть завершена лишь в течение недопустимо длительного времени. Если вы предусмотрите корректную обработку ситуаций подобного рода, то ваше приложение от этого значительно выиграет, и будет не только отлично работать, но и лучше восприниматься пользователями. Пользователи будут благодарны вам или, по крайней мере, не будут честить ваше имя при появлении сообщения наподобие “соединение с сетью отсутствует”.

Уровни абстракции программной модели

Как и в случае локальных данных устройства, при работе с сетевыми ресурсами также используются несколько уровней абстракции. Так, в .NET Compact Framework предлагаются следующие уровни API-интерфейсов для работы с сетевыми данными:

- Сокеты, использующие потоки.
- Запросы/ответы HTTP.
- Механизм SOAP.

Каждый из этих уровней абстракции, перечисленных в порядке их повышения, последовательно предлагает все более удобную и надежно протестированную инфра-

структуру. Как и в случае локального обмена данными на устройствах, вы всегда должны выбирать наивысший из приемлемых для вас уровней абстракции и переходить к более низким уровням лишь тогда, когда убеждены, что более высокие уровни не в состоянии удовлетворить ваши запросы. Использование более низкоуровневых программных моделей означает более непосредственный контроль над тем, что происходит, и обеспечивает максимальную гибкость, но это достается за счет увеличения сложности кода, уменьшения возможностей переноса приложения на другие платформы и необходимости выполнения более тщательного тестирования. Вы также должны знать о том, что даже нижайшие уровни абстракции не предоставят вам возможности полного контроля; ваше приложение всегда должно будет каким-то образом справляться со сбоями в сети, а на низких уровнях абстракции это часто сделать сложнее. Должны же иметься веские причины того, что во всех случаях, кроме самых специальных низкоуровневых задач, разработчики вместо языка ассемблера используют языки более высокого уровня, и точно так же должны рассуждать и вы, выбирая коммуникационную модель для своего приложения. Выбору более высоких уровней абстракции следует отдавать предпочтение почти во всех случаях.

При необходимости вернитесь к шагам 0, 1, 2 и 3

Современная разработка является итеративным процессом. Это особенно справедливо в отношении разработки приложений для мобильных устройств. Исходя из своей квалификации и опыта, вы принимаете первоначальные решения, которые кажутся вам наилучшими, и при необходимости впоследствии пересматриваете их. Наличие опыта проектирования приложений для мобильных устройств поможет вам принять лучшие решения, но никогда не сделает их идеальными. Всегда вкрадется какая-нибудь непредвиденная проблема, которая вынудит вас заново пересмотреть свой проект и внести в него небольшие уточнения или, как это часто случается, радикальные изменения. Значительное место в оставшейся части книги отводится тому, чтобы научить вас с самого начала принимать наилучшие решения и одновременно привить необходимые технические навыки, которые позволят оставить открытыми двери для пересмотра проекта, если это потребуется.

Хороший проект мобильного программного обеспечения должен в обязательном порядке предусматривать обнаружение проблем, появления которых вам не избежать, уже на самых ранних стадиях разработки и ориентировать на создание кода, обладающего достаточно модульной и гибкой структурой, чтобы при необходимости в него можно было внести изменения, не рискуя безнадежно запутаться.

Приступив к проектированию и тестированию коммуникационной модели, вы можете обнаружить, что в проекте были допущены существенные просчеты. Может оказаться, что расходуемый объем памяти слишком велик, а это означает, что либо количество данных, одновременно хранящихся в памяти, слишком большое, либо их хранение организовано недостаточно эффективно. Вероятнее всего, изменение модели данных окажет влияние на коммуникационную модель, управляющую тем, как именно будет осуществляться обмен данными с долговременным хранилищем. В свою очередь, внесение изменений в модель данных и коммуникационную модель может повлечь за собой необходимость пересмотра пользовательского интерфейса. Возможно, при внесении этих изменений вам придется учесть тот факт, что количество данных, одновременно находящихся в памяти в любой момент времени, превышает то, кото-

рое допускалось ранее, или же что для обеспечения желаемой степени интерактивности пользовательского интерфейса способ извлечения данных должен быть изменен по сравнению с первоначально предполагаемым. Наконец, в ходе этого процесса вы можете обнаружить, что некоторые из целей приложения, предусмотренные проектом, должны быть изменены. Возможен и другой вариант, когда в результате использования и тестирования приложения обнаруживается, что состав его проектных целей должен быть расширен. Современная разработка приложений носит итеративный характер. Поскольку мобильные устройства представляют собой новый и еще не до конца исследованный мир, они потребуют еще большего итерирования.

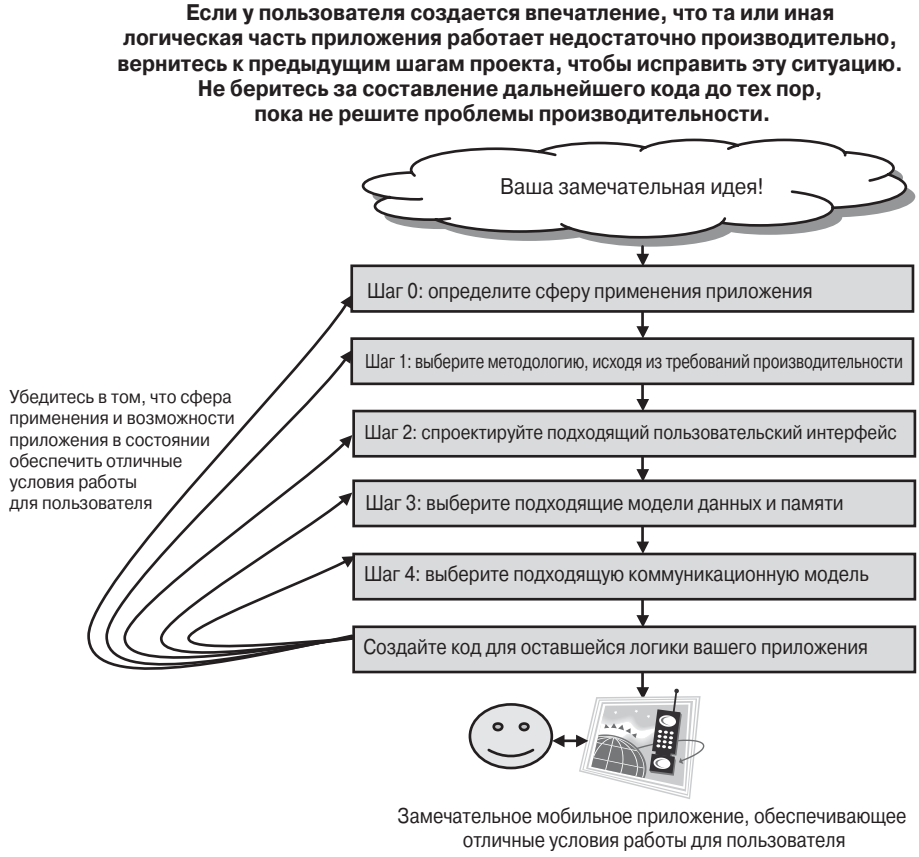

Рис. 4.5. *Пересмотрите предыдущие шаги проекта, если это необходимо для устранения проблем производительности*

В попытках согласовать между собой результаты, получаемые на различных стадиях разработки, легко потеряться где-то посередине и начать писать код, значительная доля которого, связывающая его отдельные части между собой, будет походить на запутанный клубок спагетти, а необходимые проектные изменения будут выглядеть как заплаты. Вы должны избегать подобного соблазна и использовать собственные контрольные этапы, которые помогут вам сохранить целостность процесса проектирова-

ния. В явном виде определите критерии завершения таких этапов, что позволит рационализировать процесс проектирования. По завершении контрольного этапа разработки вы должны дать ответы на следующие вопросы:

- Остаются ли в силе намеченные вами первоначальные ключевые сценарии и набор возможностей приложения или они должны быть переопределены?
- Способен ли пользовательский интерфейс точно представлять указанные сценарии, и дает ли он пользователям возможность выполнять наиболее распространенные операции при минимальном количестве манипуляций вручную? Сохраняет ли пользовательский интерфейс способность в любой момент реагировать на действия пользователя? Соответствует ли пользовательский интерфейс форм-фактору устройства, являющегося целевым?
- Работоспособна ли базовая модель данных, в соответствии с которой объекты загружаются в память и выгружаются из нее? Обеспечивает ли она масштабирование до тех реальных объемов данных, с которыми столкнутся ваши пользователи? Будет ли эта модель вести себя достаточно стабильно в процессе того, как пользователь своими действиями заставит приложение побывать во всех возможные состояниях, запросит дополнительные данные или запустит приложение, предоставив ему возможность выполняться непрерывно в течение многих недель?
- Удовлетворяет ли коммуникационная модель мобильного приложения вашим требованиям? Используете ли вы наивысший из уровней абстракции API-интерфейсов и форматов файлов, который является для вас приемлемым?

В основе всех ваших проектных решений должна лежать философия, в которой во главу угла ставится достижение высокой производительности приложения. Если какой-либо из пунктов проекта вашего приложения не в состоянии обеспечить выполнение этого требования, *прекратите дальнейшее написание кода*! *Прекратите написание кода*! *Прекратите написание кода*! Затем немедленно займитесь выяснением того, какие именно факторы приводят к снижению производительности, и, прежде чем двигаться дальше, решите связанные с этим проблемы. Это замечание играет важную роль на любой стадии проекта, но его необходимо обязательно учитывать на завершающей стадии и во всех контрольных точках проекта, которые вы для него определили. По мере приближения процесса разработки приложения к своему завершению внесение изменений должно требоваться реже, да и делать это уже будет труднее, поскольку между написанными к этому времени модулями кода установятся всевозможные зависимости. Предусмотрите в своих планах, что на ранних стадиях проекта вы будете множество раз возвращаться к критическому пересмотру принятых ранее решений и вносить изменения, тогда как по мере прохождения вашим проектом контрольных точек и приближения кода к его окончательному виду необходимость в таких изменениях будет постепенно уменьшаться.

Шаг 5: пакетирование приложения для его установки

В случае мобильных устройств процесс пакетирования и установки приложения часто называют *инициализацией*, или *подготовкой к работе*. Этот шаг должен быть отнесен к категориям, числящимся под рубрикой “последние по порядку, но не менее важ-

ные", поскольку, как и все остальные шаги, его корректное выполнение также требует применения итеративного подхода. Следует обязательно продумать, какие компоненты должны быть развернуты, чтобы ваше приложение могло выполняться на целевых устройствах, каким образом пользователи должны это осуществлять и какие способы будут использоваться впоследствии для обновления приложения. Ниже приведен перечень вопросов, давая ответы на которые вам будет легче выработать стратегию инициализации вашего мобильного приложения:

- *Какие модули развертываются вместе с приложением?* Представляет ли собой приложение единственный двоичный исполняемый файл или имеются дополнительные файлы данных, устанавливаемые вместе с ним, например, изображения, текстовые файлы или файлы баз данных? Должен ли и может ли любой из этих файлов быть включен в двоичный файл приложения в качестве ресурса для упрощения процесса развертывания? Требуются ли приложению другие библиотеки времени выполнения или компоненты, подлежащие установке вместе с ним на устройстве?
- *Какая процедура используется для развертывания приложения на устройстве?* Устанавливается ли приложение на самом устройстве через сетевое соединение? Устанавливается ли оно через настольный компьютер, с которым устройство связано кабелем? Устанавливается ли оно с карты, вставляемой в устройство? Устанавливается ли приложение с одного устройства на другое посредством инфракрасного порта, облегчающего распространение приложения между равноправными устройствами?
- *Кто будет осуществлять развертывание приложения?* Будут ли это делать конечные пользователи? Знакомы ли конечные пользователи с целевыми устройствами и процедурами инсталляции? Будут ли организации развертывать приложения на устройствах своих сотрудников? Будет ли приложение развертываться оператором мобильного телефона или загружаться из Web?
- *Как будут создаваться новые версии приложения?* Будет ли приложение само проверять необходимость своего обновления за счет использования сетевого ресурса? Будет ли сервер сканировать устройства, чтобы определить, нуждаются ли они в установке новой версии? Будут ли новые версии приложения устанавливаться при помещении устройства в лоток ПК?
- *Могут ли возникнуть проблемы защиты приложения?* При широком развертывании любого приложения всегда возникают вопросы, связанные с его защитой, которые обязательно следует рассмотреть. Должно ли приложение снабжаться цифровой подписью? Должны ли данные, с которыми работает приложение, быть тем или иным способом защищены? Что может произойти с данными, если устройство, на котором хранятся данные, будет утеряно или украдено?

Поскольку существует бесчисленное множество всевозможных мобильных устройств и разработанных для них сетевых моделей, ни на один из поставленных выше вопросов невозможно дать единственный правильный ответ. Какое решение будет наилучшим, зависит от типа устройства, способа его подключения к сети, опытности пользователя и политик, которые сетевые операторы могут использовать для контроля доступа к устройству.

План инициализации приложения следует записать на бумаге и добавить в список сценариев. Также крайне желательно включать разработку и проверку корректности процедуры пакетирования и установки приложения в реальных условиях в состав контрольных этапов начальных стадий проекта. Современные инструментальные среды разработки могут прекрасно справляться с задачей установки приложения на устройстве в целях тестирования, но при самостоятельной установке приложения конечными пользователями среда разработки для этих целей обычно не применяется. Как показывает практика, развертывание приложения конечными пользователями всегда является проблемой.

Резюме

Хорошо известна старая, проверенная временем истина: "Лучше хорошо выполнить посредственный план, чем иметь самый хороший, детально проработанный план, который так и остается нереализованным". Эта истина применима и к разработке современного программного обеспечения. Что толку от того, что у вас имеются разработанные по всем правилам план и методология, если они недостаточно гибки, чтобы их можно было приспособить к реалиям итеративного характера получения конечного результата. "Для разных проектов разработки требуются разные уровни строгости и опыта" — вот лучшее руководство к пониманию того, какая степень формализации обернется наибольшей выгодой для проекта. Кроме того, следует быть постоянно готовым к любым неожиданностям. По мере того как вы будете разрабатывать и тестировать приложение, обязательно будут выявляться факторы, которые вынудят вас существенно изменять свои планы. Всегда учитывайте это и организуйте процесс разработки таким образом, чтобы он позволял справляться с подобными ситуациями. Итак, в хронологическом порядке:

1. *Вооружитесь определенной методологией разработки и строго придерживайтесь ее.* Одна из этих методологий была представлена в общих чертах в этой главе и рассматривается более подробно в последующих главах. Эта методология хорошо приспособлена для мобильных устройств. Если окажется, что ваши потребности несколько отличаются, можете изменить ее, но в любом случае вы должны действовать в соответствии с определенной методологией.
2. *Подготовьте единственный основной документ проекта, который определяет ваши цели и позволяет отслеживать степень выполнения проекта.* Уровень формализации этого документа будет зависеть от того, что собой представляет ваша организация. Если вы — индивидуальный разработчик, работающий автономно, то этот документ будет просто помогать вам отслеживать разработку высокоуровневых задач и сценариев, определять ваши контрольные точки в процессе решения этих задач, а также отслеживать невыполненные пункты, подлежащие дополнительному анализу; объем такого документа не должен превышать нескольких страниц. В случае если у вас крупная организация, этот документ может представлять собой официальный контракт между вами и другой организацией, в котором подробно описывается, что именно должно быть выполнено, и какие проектные решения были приняты, а также детализированы контрольные этапы проекта и его состояние на протяжении всего периода разработки. В соответствии с этим документом и должна осуществляться разработка, и вы все время

должны поддерживать его "актуальность". Пусть это будет единственный документ, который представляет задачи вашего проекта и позволяет объективно оценивать этапы его выполнения.

3. *Идентифицируйте контрольные точки, для которых четко определены критерии их выполнения.* Это лучший способ помочь вам и участникам группы разработчиков быть честными перед самими собой при оценке результатов вашей работы. Набор четко определенных контрольных точек, которые позволяют оценивать степень завершения разработки приложения и решения поставленных задач, — бесценная вещь, независимо от того, работаете ли вы один или в составе большой организации. Достоинства контрольных точек (этапов) просто невозможно переоценить. Если вы не совсем хорошо представляете себе, какое количество контрольных точек является оптимальным, рекомендую начать с пяти. Если вам не удается разбить свой проект на пять поддающихся оценке шагов, то либо ваш проект настолько тривиален, что его можно выполнить буквально за пару дней, либо вы были недостаточно старательны при определении этапов. Пять шагов — это разумное количество для начала, которое позволит достаточно подробно анализировать степень вашего прогресса. В случае более крупных проектов может оказаться желательным дополнительное разбиение этих контрольных точек на ряд более мелких; в случае менее крупных проектов вы можете обойтись четырьмя контрольными точками, но старайтесь не использовать меньшее их количество. Введение контрольных точек выполняет три основные функции: 1) они позволяют вам объективно оценивать продвижение к намеченным целям; 2) они обеспечивают возможность пересмотра намеченных целей и контрольных точек в будущем с учетом всего нового, что вы узнаете в процессе работы над проектом; 3) завершение контрольной точки дает формальный повод критически проанализировать проведенную работу, подчистить код и скорректировать проект с учетом любых подходящих рационализаторских предложений. Без введения контрольных точек и доводки их конечных результатов существует риск того, что вы будете непрестанно заняты одним только написанием кода, и все ваши решения относительно этого будут носить исключительно тактический характер. Полученный код будет представлять собой сплошное "спагетти", сопровождение которого будет весьма затруднительным. Даты контрольных точек можно устанавливать, а можно и не устанавливать; чем многочисленнее группа разработчиков, тем бо́льшую значимость приобретает установление контрольных дат. Самое главное, чтобы все участники приходили к финишной линии текущей контрольной точки вместе, и только после этого переходили к выполнению очередной контрольной точки. Контрольные точки — ваши друзья, используйте их!

4. *Руководствуйтесь требованиями производительности.* В случае приложений для мобильных устройств из множества критериев безраздельное первенство принадлежит удобству использования, а главное влияние на этот аспект приложения оказывает его производительность. По мере продвижения работы над проектом проблемы производительности, решение которых оставлено "на потом", будут только усугубляться, и это является непреложной истиной. Используемые при построении мобильных приложений модели характеризуются ограниченностью доступных объемов памяти и ресурсов, что существенно отличает их от

более гибких моделей, используемых в случае настольных компьютеров. Установите для себя жесткие правила во всем, что связано с обеспечением высокой производительности приложения и его интерактивных возможностей, и удерживайте весь процесс разработки в русле этих требований. Свяжите решение проблем производительности с критериями завершения контрольных точек. Именно производительность будет окончательно решать, на что способно ваше мобильное приложение. Привлекательной серебряной блесткой на фоне этой темной тучи, витающей над всем процессом разработки, является то, что сформулированная выше цель вполне достижима. Это становится возможным при наличии строго продуманного плана работ, соблюдения определенной дисциплины в процессе разработки и творческого подхода к решению возникающих проблем. Разработчики, умонастроение которых подчинено привычкам, выработанным при создании приложений для настольных компьютеров, столкнувшись с проблемами производительности, могут решить, что в невозможности реализовать то, что им хотелось, виноваты собственно устройства. Однако, по большому счету, в подобных случаях все объясняется недостатком воображения у разработчика. При хорошо продуманном пользовательском интерфейсе, правильно подобранных моделях данных и памяти, а также разумном выборе коммуникационной модели существует мало задач, решение которых не может быть обеспечено современными приложениями для мобильных устройств.