Глава 3

Разработка приложений на Visual Basic 2005

В этой главе...

- подробное рассмотрение платформы .NET Framework;
- планирование работы;
- реализация плана;
- описание программного обеспечения.

Прежде чем изучать многообразие доступных в Visual Studio проектов, простоту Visual Basic и мощь .NET Framework, читателю необходимо узнать, как проектируется программное обеспечение. Под *проектированием* здесь понимается планирование работы. Обычно создание программного обеспечения сравнивают с постройкой здания. Вы бы наняли конструктора, который собирается строить дом без чертежей? Конечно, нет. Аналогично, не следует думать, что можно написать программное обеспечение без проектирования.

Читателю предстоит выяснить, как создавать программное обеспечение с самого начала. В этой единственной посвященной проектированию главе будет показано, как .NET Framework позволяет решить эту задачу. Кроме того, здесь рассматривается структура .NET Framework и то, как .NET Framework взаимодействует с Visual Basic и Visual Studio.

Затем рассматривается фундамент, на котором построена платформа .NET Framework, абстрактные понятия, делающие ее такой простой в использовании, а также некоторые конкретные примеры ее использования.

В этой главе также рассматривается планирование разработки программного обеспечения. Верите вы в это или нет, но существуют общепринятые, структурированные способы разработки программного обеспечения. Разработчик, придерживающийся этой структуры, получает великолепный способ описать свои проектные планы на бумаге. В данной главе обсуждается проектирование программного обеспечения, реализация которого рассматривается в Части II этой книги.

Наконец, я расскажу, как описать программное обеспечение с точки зрения чтения и написания проекта. После изучения этой главы читатели смогут планировать реальные программные проекты.

Упрощение программного обеспечения с помощью .NET Framework

.NET как концепция представляет сбой разработанную Microsoft библиотеку связанного программного обеспечения, которое позволяет людям и используемым ими системам и устройствам связываться с необходимой им информацией. .NET Framework — среда разработки, которая позволяет это делать с точки зрения Visual Basic.

Visual Basic — это просто часть .NET Framework. Как показано на рис. 3.1, Visual Basic используется только для того, чтобы написать клиентское и серверное программное обеспечение, а также программы, обеспечивающие взаимодействие сервера и клиента.

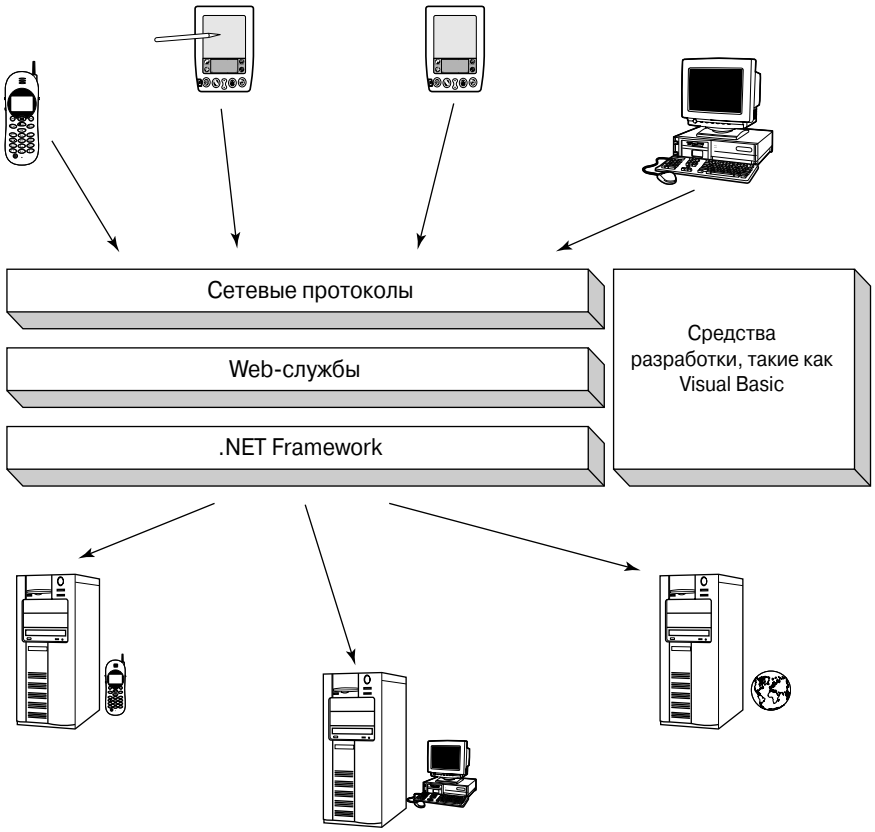

Рис. 3.1. Структура приложения

Правильно спроектированное программное обеспечение включает в себя описанные ниже уровни.

- ✓ Клиенты в среде .NET — это такие устройства, как сотовые телефоны и PDA, персональные компьютеры под управлением Windows или Web-браузеры в любой операционной системе.
- ✓ Серверы в .NET обычно работают под управлением Windows Server и SQL Server. В .NET серверная платформа имеет гораздо меньшую гибкость, чем клиентская платформа. Есть и другие варианты, такие как базы данных Oracle. Иногда используются такие серверы, как BizTalk или службы SharePoint. Вообще, все очень просто — серверы нужны для того, чтобы обслуживать клиентов.
- ✓ В середине находятся XML Web-службы. XML Web-службы представляют кросс-платформенную стратегию передачи информации от серверов к клиентам, между клиентами или даже между самими серверами.

Средствами разработки, представленными на рис. 3.1, являются Visual Basic и Visual Studio. Visual Basic — язык, а Visual Studio — инструмент. Третьим фрагментом головоломки является план — тип проекта. План является главной темой этой главы.

В верхней части рис. 3.2 показаны все структуры, составляющие блок “Средства разработки” на рис. 3.1. VB (Visual Basic) и является одной из таких структур.

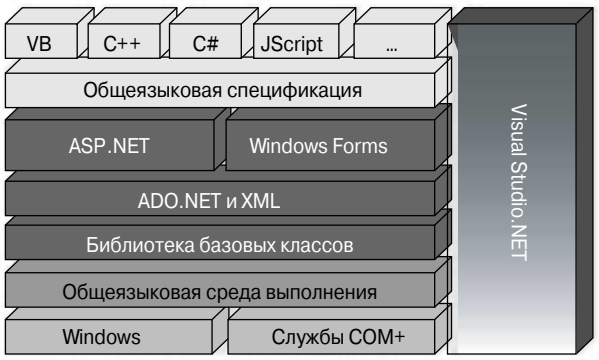

Рис. 3.2. .NET Framework

Также весьма важно то, как VB взаимодействует с другими частями, представленными на диаграмме, то есть то, что делает ваше программное обеспечение. Программы используют службы, представленные в языке платформой .NET Framework через инструменты. Это взаимодействие является ключевым фактором для всего — вокруг него должно быть сосредоточено планирование. Как извлечь преимущества из взаимодействия компонентов .NET Framework? Для этого и разрабатывается программное обеспечение.

Прежде чем перейти к планированию, необходимо знать, чем .NET Framework, как часть Visual Basic, может помочь разработчику. Одна из сложнейших частей планирования программного обеспечения заключается в том, чтобы определить, что программа должна делать для пользователя и что уже сделано службами .NET Framework. В нескольких последующих разделах рассматриваются возможности, которые .NET Framework предоставляет разработчику.

Обращение к операционной системе

Главным компонентом .NET Framework является библиотека базовых классов (Base Class Library, BCL), которая обеспечивает доступ к функциям операционной системы и таким службам, как графика и базы данных. Все остальное реализуется множеством вспомогательных компонентов платформы, но доступ к операционной системе — важнейший фактор для Visual Basic. Почему? Раньше, чтобы получить доступ к этим службам, программисты на Visual Basic вынуждены были прибегать к разным ухищрениям.

Путь из Visual Basic в операционную систему долгий и извилистый. Объект `My` сокращает этот путь. Кроме того, этот объект является замечательным примером того, как .NET Framework может помочь VB-программисту делать свою работу.

Объект `My` предоставляет доступ к компьютеру через средства операционной системы. Объект `My.Computer` позволяет программе легко взаимодействовать со всеми компонентами компьютера:

- ✓ клавиатурой и мышью;
- ✓ принтерами;
- ✓ аудио и видео подсистемами;
- ✓ буфером обмена;
- ✓ часами;
- ✓ файловой системой.

Ниже представлены некоторые распространенные задачи, которые можно легко выполнить с помощью объекта `My`:

- ✓ загрузка и выгрузка файлов;
- ✓ чтение, запись и удаление данных из буфера обмена;
- ✓ управление подключением компьютера к Интернету.

Эти задачи довольно сложны для большинства промышленных языков программирования, но в Visual Basic, использующем .NET Framework, они решаются значительно проще.

Более того, в семействе `My` есть еще два важнейших объекта. Объект `My.Application` помогает программам получать информацию о среде, в которой они выполняются. Объект `My.User` помогает собрать информацию о пользователе, который зарегистрировался в системе, например, его имя и e-mail-адрес.

Пиктограммой “Объект My” в этой книге отмечены рекомендации по эффективному использованию объекта `My` в разработке приложений.

Интеграция серверов и служб

В средней секции рис. 3.2 имеется четыре блока, причем два из них — это компоненты, ориентированные на пользовательский интерфейс, а два других — компоненты, ориентированные на службы. ASP.NET и Windows Forms — это компоненты, ориентированные на пользовательский интерфейс, они рассматриваются ниже в разделе “Взаимодействие с пользователем”. ADO.NET и компоненты BCL важны отчасти потому, что они помогают интегрировать такие серверы, как базы данных и службы вроде BizTalk.

Компонент ADO.NET содержит основной сервер, который необходимо интегрировать, — сервер данных. Чаще всего программы на Visual Basic должны взаимодействовать с базами данных, такими как Microsoft SQL Server 2005. Программы для бизнеса часто должны передавать информацию от пользователя пользователю, и такая информация, как правило, хранится в базах данных.

ADO.NET позволяет, используя минимальное количество кода, выбрать данные из базы, показать их пользователям, выполнить заданную пользователями обработку данных и обновить базу. То есть вы можете разрабатывать бизнес-логику приложения, не заботясь о том, как работает само соединение с базой данных. Соединения с базами данных и технология ADO.NET рассматриваются в главе 15.

Кроме серверов баз данных, существуют и другие типы серверов. Частично работа библиотеки базовых классов заключается в том, чтобы предоставить программисту возможность подключаться к этим серверам без необходимости писать код подключения. Помимо организации соединений, которую обеспечивает для приложений библиотека BCL, внутри нее также имеется множество служб, доступ к которым осуществляется достаточно просто за счет мощи BCL. Ниже кратко описаны эти службы.

- ✓ **Enterprise Services (Службы предприятия):** инструменты, которые нужны очень крупным приложениям, например, транзакция и активизация, обеспечиваемые службой компонентов (Component Service).
- ✓ **IO (Ввод-вывод):** доступ к файловой системе, дискам и хранение данных на серверах с различными операционными системами. Ввод-вывод рассматривается в главе 16.
- ✓ **Messaging (Сообщения):** использование службы очередей сообщений в Windows. (Следует отметить, что здесь имеются в виду не мгновенные сообщения пользователей, а сообщения между программами в Windows). Этот вид сообщений используется приложениями для передачи и получения сообщений о взаимодействии пользователей с данными.
- ✓ **Management (Управление):** доступ к инструментарию управления Windows (Windows Management Instrumentation), службы которого позволяют оценить состояние сервера.
- ✓ **Net (Сеть):** сеть и Интернет. Все Web-сайты и e-mail-серверы доступны благодаря семейству Net в библиотеке BCL, которое рассматривается в главе 17.
- ✓ **Drawing (Рисование):** истинное искусство — тяжелый труд. Библиотека BCL позволяет воспользоваться ее мощью, облегчая доступ к множеству графических инструментов Windows, которые называются библиотекой GDI+. Рисование рассматривается в главе 18.

Взаимодействие с пользователем

В центральной секции на рис. 3.2 показаны еще два блока — ASP.NET и Windows Forms, которые позволяют реализовать взаимодействие программы с пользователем.

Выше уже было сказано, что разработка программ в Visual Basic состоит из трех частей. Первая часть — язык. Вторая часть — инструмент, Visual Studio 2005, который обсуждается в главе 2.

Третья и последняя часть головоломки — тип проекта, или платформа, которая контролируется ASP.NET или Windows Forms. Технология ASP.NET предназначена для разработки Web-страниц, мобильных Web-приложений и XML Web-служб; Windows Forms — для разработки Windows-приложений, консольных программ и приложений для интеллектуальных устройств.

Более подробная информация о взаимодействии с пользователем приведена в главах 4 и 5.

Сравнение абстрактных идей с объектами реального мира

Вы можете потратить огромное количество времени, читая об абстрактных и конкретных понятиях, связанных с работой в .NET. Так, в статьях или документации в Web можно встретить множество упоминаний о *классах* и *объектах*.

Классы

Классы — философское понятие. Это емкости, которые можно чем-нибудь наполнить, каркасы или скелеты, на которые можно надеть оболочку, полосы почвы, ожидающие посадки растений. Они абстрактны и не конкретны.

Класс является определением чего-нибудь со списком того, что можно с этим сделать, что известно об этом и что это может делать. Если “экземпляр класса” не создан, то класс можно сравнить с множеством клеток для животных, которых там пока нет.

Объекты

Объекты — существующие в приложении конкретные элементы. Объекты — это то, во что классы превращаются, когда вырастают. Когда создается экземпляр класса, класс "одевается" в оболочку и становится объектом.

Когда вы определяете что-нибудь в приложении, вы создаете класс. Фраза "Дом имеет `Цвет` и `ПараднуюДверь`" была бы примером класса, просто его определением. *Экземпляр (instance)* класса `Дом`, созданный и наполненный данными, становится конкретным домом, объектом `МойДом`. На основе чертежа, которым является класс, можно построить сколько угодно домов, потому что каждому из них выделено собственное пространство в памяти, где сохраняется его собственная информация, называемая *состоянием (state)*.

Разработка классов рассматривается в главе 6, однако классы и объекты упоминаются почти в каждой главе этой книги и повсеместно используются в Visual Basic. Все в .NET является объектом, конкретизированным существованием самого приложения. Но, когда Microsoft разрабатывала объекты, они были просто классами.

Планирование проекта на основе жизненного цикла

Подготовка к созданию нового продукта состоит из двух отдельных этапов, планирования и проектирования. *Планирование* состоит из определения проекта и сбора проектных требований. *Проектирование* состоит из описания экранов и логики, которые соответствуют требованиям, и определения способов проверки их правильности.

Вместо того чтобы писать о том, как придерживаться этих предписаний, я расскажу о планировании и проектировании проекта, который будет реализовываться во второй части книги. В качестве примера проекта создается программа, вычисляющая даты. Что делает программа Date Calculator и как она работает, вы выясните в ходе жизненного цикла проекта.

Жизненный цикл проекта представляет собой процесс, который лучше всего демонстрируется графиком Ганта (см. рис. 3.3).

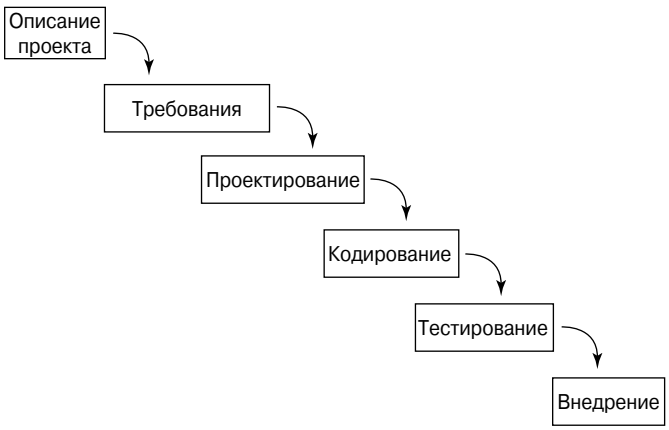

Рис. 3.3. Жизненный цикл проекта

Проекты должны выполняться с использованием этого процесса. Те, кто уже какое-то время программируют на Visual Basic, вероятно, заметили, что многие программисты, использующие

другие языки, иногда пренебрежительно относятся к Visual Basic. Отчасти это связано с тем, что в VB очень легко писать программы, не планируя их вообще, а это не правильно.

Следует отметить, что описываемый здесь жизненный цикл проекта является только одним из многих процессов разработки. Вы, скорее всего, уже сталкивались со многими промышленными терминами, которые представляют различные стороны одной базовой системы понятий. Независимо от того, как вы называете процесс разработки, планирование и проектирование весьма важны. Об этом следует помнить.

Чтобы написать хорошее приложение, сначала необходимо создать план. Несмотря на то, что этапы плана могут частично перекрываться, каждый этап должен быть завершен. Даже в небольших проектах время, потраченное на проектирование, окупается. Дефекты, которые обнаруживаются во время планирования, в показателях времени обходятся почти в десять раз дешевле, чем дефекты, которые обнаруживаются и устраняются во время разработки и тестирования.

Первые три этапа цикла (см. рис. 3.3) ставят перед разработчиком вопросы, проясняющие проектные требования. В случае планирования приложения, которое разрабатывается во второй части книги, эти вопросы могут быть сформулированы так, как показано ниже.

- ✓ **Рамки проекта:** нужны ли в программе расчета дат другие вычислительные возможности? Для какой платформы предназначена программа? Должна ли программа быть интернациональной?
- ✓ **Требования:** что будет вычислять программа? Как пользователь будет вводить даты? Какие результаты должна выдавать программа?
- ✓ **Дизайн:** как программа будет вычислять даты? Какие элементы пользовательского интерфейса будут лучше всего показывать данные? Как будут запускаться вычисления в программе? Как будет выглядеть экран?

Определение рамок проекта

Определение рамок проекта является наиболее важной частью процесса проектирования, поскольку от этого зависит, что именно будет делать приложение. Если кто-нибудь спросит вас о том, что делает ваше приложение, вы должны знать, что ответить. Рамки проекта определяют то, чего приложение не будет делать; это, возможно, даже еще важнее. С этим связан термин “выходит за рамки проекта”.

Попытайтесь составить определение проекта из 101 слова или даже с помощью еще меньшего количества слов. Это позволяет сохранить небольшой объем проекта, потому что вам придется взвешивать каждое слово.

Так как программа Date Calculator довольно проста, для ее описания может и не понадобиться 101 слово. Маркированные списки — удобный способ описания рамок проектов.

- ✓ Date Calculator — приложение, которое определяет разницу между двумя датами, заданными в формате США.
- ✓ Программа выполняется на Web-странице или Windows-компьютере, либо как функция в любом другом приложении.

Это описание определяет проектируемое приложение. Если пользователь скажет, что в приложении, которое называется Date Calculator, должна быть функция сложения двух чисел, вы можете ответить, что эта функция выходит за рамки проекта. С другой стороны, если это ожидаемое требование, то рамки проекта необходимо изменить, для чего придется вернуться к этапу планирования.

Сбор требований

Требования представляют собой специфические правила, регламентирующие приложение. Требования можно представить как проблемы, которые должны быть решены на этапе проектирования. Для программы Date Calculator эти требования довольно просты, их перечень приведен ниже.

- ✓ Программа принимает начальную дату (в формате США), `startDate` и целочисленный интервал дат, `span`, а возвращает дату `endDate`, которая наступает через `span` дней после даты `startDate`.
- ✓ Кроме того, Date Calculator может принимать две даты (в формате США), `startDate` и `endDate`, и возвращать целое количество дней между ними, т.е. `span`.
- ✓ Должна быть предусмотрена возможность реализовать Date Calculator как:
  - Windows-приложение в виде исполняемого файла;
  - Web-приложение, выполняемое в клиент-серверной среде;
  - повторно используемый компонент в Windows;
  - XML Web-службу.
- ✓ Если возможно, то результаты пяти предыдущих вычислений будут сохраняться приложением от сеанса к сеансу.

Требования типа “если возможно” встречаются довольно часто. По сути, они состоят из функций, которые могут вписываться или не вписываться в бюджет. Такие требования лучше включать в конец списка.

Выше перечислены все сведения, которые нужны для описания функциональности приложения. Эта информация должна быть оформлена в виде документа, который уместно называть *перечнем требований (requirements document)*. Этот документ может быть Word-файлом, текстовым файлом, листком для записей или даже обычной салфеткой. Создание и использование *перечня требований* позволяет быть уверенным, что завершенное приложение делает то, что и предполагалось.

Каждый пункт перечня требований должен быть раскрыт в отдельном пункте следующего документа — плане проекта (design document). Можно пронумеровать пункты перечня требований и плана проекта, чтобы гарантировать, что каждое требование отражено в проекте.

Когда требования утверждены, наступает время описать программное обеспечение с технической точки зрения; это завершающий этап проектирования. В следующем разделе рассматриваются такие этапы, как рисование пользовательского интерфейса и определение логики.

Дизайн Date Calculator

В нижней части графика проектного цикла отмечены в основном технические вопросы. Этапы дизайна, написания кода, тестирования и внедрения в крупных компаниях обычно реализуются разработчиками, а не бизнес-аналитиками. Индивидуальным разработчикам приходится делать все это самостоятельно.

Необходимо тщательно и всесторонне описать программное обеспечение, чтобы можно было передать документ начинающему программисту для работы. В случае Date Calculator необходимо рассмотреть три главных пункта. Сосредоточение усилий на перечисленных ниже логических разделах облегчает разработку большинства программ.

- ✓ Проектирование данных.
- ✓ Рисование пользовательского интерфейса.
- ✓ Схематическое изображение соединений между бизнес-уровнем и уровнем данных.

Хранение данных

Программа Date Calculator сохраняет собранные и вычисленные ею данные. Если вы внимательно читали требования в разделе “Сбор требований”, то, вероятно, заметили, что существует только три значения:

- ✓ первое — `startDate`;
- ✓ второе — `endDate`;
- ✓ количество дней между `startDate` и `endDate`, `span`.

Кроме этого, можно сохранять следующие данные:

- ✓ дату, вычисленную в ходе предыдущего сеанса работы программы;
- ✓ имя пользователя, запускавшего Date Calculator в предыдущий раз;
- ✓ некоторую ссылку для поиска.

Всю эту информацию можно хранить в одной структуре данных, которую можно назвать `Calculations` (вычисления). Обычно такие данные описываются с помощью сетки, аналогичной таблице на рис. 3.4. Эта диаграмма называется диаграммой связи сущностей (Entity Relationship, ER), т.к. если бы существовало несколько сущностей (структур данных), то связи были бы показаны в виде линий между ячейками. Так обычно показываются базы данных.

Calculations	
PK	**CalculationId**
	StartDate Span EndDate **DateRan** User

Рис. 3.4 Диаграмма связи сущностей для структуры Calculations

Понятие многоуровневой конструкции

Многоуровневой называется система, имеющая уровни представления, бизнес-логики и доступа к данным, которые физически выполняются на различных платформах, и по крайней мере один уровень распределен среди этих платформ. Многоуровневая архитектура идеально подходит для Web-приложений, потому что в них уровень представления распределен между Web-браузером и Web-сервером, а бизнес-логика и компоненты доступа к данным могут быть разделены — почти как в клиент-серверном приложении — между брокером объектных запросов (Object Request Broker) и системой управления базами данных (Database Management System).

При проектировании крупной системы, сначала лучше определить базу данных, или уровень данных (Data Layer). Затем разработать пользовательский интерфейс, или уровень представления (Presentation Layer). И, наконец, связать эти уровни вместе, используя уровень бизнес-логики (Business Logic Layer).

Многоуровневая система предоставляет два преимущества. Модульность хорошей многоуровневой конструкции позволяет удалять или заменять определенный компонент системы, не нарушая функциональности остальной части приложения. Кроме того, отделение бизнес-логики от базы данных позволяет обеспечить балансировку нагрузки, безопасность и общую стабильность работы в системах с высоким уровнем отклика. Практический результат заключается в том, что многоуровневые транзакционные системы заменяют большое количество COBOL-кода, который управлял мировой экономикой. Если вы хотите внести свой вклад в этот процесс, вам следует понимать многоуровневые системы.

Используя ER-диаграмму, можно понять тип информации, которую будет обрабатывать приложение, а это весьма полезно. Три из указанных блоков информации, или *полей*, представляют пользовательскую информацию и три — системную информацию. В следующем разделе будет показано, как разработать макет экрана, в котором эти поля будут использоваться соответствующим образом.

Проектирование экранов

Сведения, собранные в разделе "Сбор требований", позволяют утверждать, что приложение должно быть мультиплатформенным. Из четырех платформ, которые необходимо разработать, только две (Windows Forms и Web Forms) имеют пользовательские интерфейсы, и эти интерфейсы очень похожи друг на друга. Для обоих пользовательских интерфейсов необходимо использовать одинаковый дизайн.

На основании требований и схемы данных необходимо создать три поля (элемента управления), которые должны быть узнаваемыми для пользователей, и разработать способ ввода информации в систему. Затем необходимо создать какой-либо элемент управления для поддержки функции "последние пять".

Я рекомендую начать с полей ввода и вывода. Понадобится три реализации этих полей. Чтобы собирать пользовательские данные, проще всего применить три текстовых поля с соответствующей подписью для каждого из них.

Здесь приходят на помощь функции взаимодействия с пользователем из библиотеки базовых классов (Base Class Library). Из документации можно выяснить, что и ASP.NET и Windows Forms содержат элемент управления, который позволяет пользователю выбирать дату из календаря. Это как раз тот случай, когда знания о том, чем система может помочь разработчику, оказываются как нельзя кстати. Многие программисты могут создавать собственные элементы управления для выбора даты, не зная, что такой элемент уже есть.

Итак, необходимо использовать два элемента управления для выбора дат, одно текстовое поле и кнопку, нажав на которую пользователь заставит программу рассчитать количество дней между двумя датами. Макет, включающий в себя все эти компоненты, выглядит аналогично окну, показанному на рис. 3.5.

Рис. 3.5. Первоначальный дизайн пользовательского интерфейса программы Date Calculator

В данный момент вы еще можете не знать, как будут работать календари, поэтому не сможете создать точную картину. Но это не страшно. Известно, что календари должны делать — предоставлять пользователю возможность выбрать дату.

Разработка пользовательского интерфейса иногда очень сильно зависит от опыта того, кто эту разработку выполняет. Я вижу, например, что та или иная конструкция будет работать и в Web и Windows-среде, но для других людей это может и не быть столь очевидным. Если вы не знакомы с Web-дизайном, но в проекте есть требования, касающиеся ASP.NET, позовите на помощь Web-разработчика. Нет ничего тяжелее, чем пытаться написать Windows-приложение, используя в качестве платформы Web.

Второе главное требование — возможность сохранять результаты пяти последних вычислений. Например, можно показывать эти результаты в виде таблицы, в которой пользователь может посмотреть сохраненные результаты. Для этого требуется ADO.NET и элементы

управления пользовательским интерфейсом. ADO.NET позволяет извлечь из базы данных пять последних результатов для пользователя, а необходимый для их просмотра элемент управления Data Grid есть как в Windows Forms, так и в проектах Web Forms.

Следовательно, нужен контейнер, в котором будет показываться начальная дата, конечная дата, интервал и искомая дата. Кроме того, пользователь должен иметь возможность щелкнуть на сохраненном результате и просмотреть его. Этот контейнер (таблица) может быть добавлена в нижнюю часть окна, как показано на рис. 3.6.

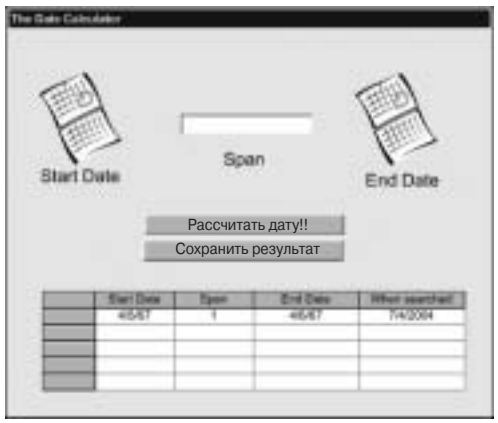

Рис. 3.6. Окончательный макет интерфейса программы Date Calculator

Вы сами пользуетесь различными программами и знаете, какие интерфейсы вам нравятся, а какие нет. Старайтесь создавать такие интерфейсы, которые вам самим хотелось бы использовать. Берите пример с популярного программного обеспечения, например, Windows, Office и др., и аналогичным образом создавайте пользовательские интерфейсы своих приложений. Несколько лет тому назад уже было доказано, что запатентовать пользовательский интерфейс нельзя.

Определение логики

Итак, известно, какие данные сохраняет приложение и как эти данные отображаются для пользователя. Последний этап заключается в том, чтобы связать пользовательский интерфейс и данные вместе. Эта связь обычно называется уровнем бизнес-логики (Business Logic Layer), и иногда этот уровень отделен от остальной части приложения и размещается на совершенно другой машине.

В Windows-разработке лучше всего просто выяснить все, что пользователь может делать в приложении, а затем написать псевдокод, который описывает эту функциональность.

Псевдокод представляет собой языконезависимые инструкции, описывающие функциональность. Псевдокод можно представить себе, как программу, написанную на английском (или на русском языке). Цель заключается в описании действий, выполняемых программой в результате взаимодействия с пользователем, с тем чтобы кто-нибудь другой, кому придется писать по псевдокоду реальную программу, читая это описание, понял, чего вы хотели добиться. В большинстве методик программирования такие описания называются *сценариями использования (use cases)*, *пользовательскими рассказами (user stories)* или *сценариями (scenarios)*.

В данном случае имеется ограниченное число действий пользователя и ограниченное число функций, которые может выполнять приложение. Ниже описана последовательность действий и предполагаемый дизайн.

- ✓ **Загрузка приложения.**
  - Присвоить переменным `startDate` и `endDate` значение текущей даты, а переменную `span` оставить пустой.
  - Предполагается, что пользователь хочет определить количество дней между двумя заданными датами.
  - Загрузить контейнер сохраненных результатов из источника данных и вывести результаты, отсортированные по дате и времени сохранения в обратном порядке.
- ✓ **Нажатие кнопки "Рассчитать дату!!!".**
  - Если в полях `startDate` и `span` есть значения, прибавить количество дней (`span`) к `startDate` и вывести полученную дату в поле `endDate`.
  - Если есть значения в полях `startDate` и `endDate`, вывести разность двух дат в поле `span`.
  - Если значения есть во всех трех полях, предположить, что пользователь хочет вычислить промежуток времени (`span`).

Предположения губят программные проекты. Никогда не делайте подобных предположений относительно дизайна. Всегда спрашивайте пользователя о том, чего он хочет. Предположения, представленные выше, используются здесь только с целью упростить пример.

- ✓ **Нажатие кнопки "Сохранить результат".**
  - Записать `startDate`, `endDate`, `span`, текущую дату, время и имя текущего пользователя в поля базы данных.
  - Обновить контейнер результатов, так чтобы последний результат оказался вверху.
- ✓ **Выбор пользователем результата в контейнере.**
  - Загрузить значения `startDate`, `endDate` и `span` в поля контейнера.
  - Заменить текущие значения в полях `startDate`, `endDate` и `span` значениями из сохраненного результата.
  - Все остальное остается без изменений.

Выше описана функциональность, которую ожидает пользователь, и эта функциональность удовлетворяет всем требованиям. Система спроектирована!

Написание плана тестирования

Прежде чем начинать писать код, следует написать план тестирования. Это просто, а кроме того, это позволяет быть уверенным, что все важные части конструкции учтены. Рассмотрим этапы написания плана.

1. **Оценить требование, которое поддерживается определенным элементом дизайна.**

   Например, приложение Date Calculator должно предоставлять пользователю возможность вводить начальную и конечную дату, а также период времени.

2. **Внести в список данный элемент дизайна.**

   Интерфейс Date Calculator должен иметь элементы управления, которые принимают вводимые пользователем данные.

3. **Описать проверку работоспособности этого элемента дизайна.**

   То, что пользователь вводит даты, гарантируется использованием элемента управления Calendar (Календарь). Но как проверить, ввел ли пользователь корректный период времени? Это должно быть целое число.

4. **Описать, что может нарушить работу элемента дизайна или вызвать ошибку.**

   "Что случится, если пользователь введет в поле `span` не целое число?" Естественно, в обычных условиях это вызовет определенную ошибку.

Возможно, потребуется изменить дизайн, чтобы гарантировать, что пользователь может ввести только число. Существует ли текстовое поле, которое позволит это сделать? Если вспомнить о поиске в Google, то станет очевидно, что такое текстовое поле существует для Windows, но не для Web. Более подробная информация о том, как Visual Basic помогает проверить пользовательский ввод, приведена в главах 4 и 5.

Просто убедитесь в наличии тестового плана, который можно передать третьему лицу, чтобы убедиться, что приложение делает то, что и предполагается. Лучше всего отдать приложение на тестирование кому-нибудь другому. Следует либо передать план для работы другому человеку, либо воспользоваться системой тестирования, такой как NUnit или Microsoft Team System.

Соблюдение плана

Теперь, когда план создан, главная трудность заключается в том, чтобы его придерживаться. Следующие советы могут помочь справиться с этой проблемой.

- ✓ **Не изобретайте велосипед.** Прежде чем писать код, поищите решения в аналогичных приложениях или примерах.
- ✓ **Читайте и исследуйте документацию.** Не старайтесь быть "продвинутым пользователем", девиз которого "Я разберусь во всем сам". Просто .NET Framework слишком велика для такого подхода. Научитесь пользоваться документацией — об этом речь идет в главе 2.
- ✓ **Пишите код так, чтобы приложение было таким, как вы хотите.** Не сдавайтесь. Если вы думаете, что можете сделать что-нибудь, старайтесь до тех пор, пока не увидите, что дело сделано. А если это не получается, то можно пересмотреть проект.
- ✓ **Пишите как можно меньше кода.** Используйте инструменты пользовательского интерфейса, предоставляемые Visual Studio. Не соглашайтесь с теми, кто думает, что весь код должен быть написан вручную.
- ✓ **Будьте последовательны.** Используйте те же имена, что и в дизайне. Определите концепции. Не используйте имена вроде *x* для обозначения численных переменных.