МОДУЛЬ 10: «ТЕСТИРОВАНИЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ»

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ И ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ

При тестировании приложений, созданных с использованием объектно-ориентированного программирования, применяются многие из методов тестирования, которые используются при процедурном программировании.

1. Проектирование тестов на уровне системы с помощью схемы.
2. Проектирование тестов на уровне системы с помощью прецедентов.
3. Проектирование поэлементных тестов для совокупности классов.
4. Проектирование поэлементных тестов для иерархии классов.

Задача заключается в составлении каких-либо парадигм проектирования тестов, чтобы можно было начать тестирование объектно-ориентированного программного обеспечения. Ограниченные знания об объектно-ориентированном программном обеспечении никоим образом не повлияют на работу тестера-профессионала.

Объектно-ориентированное программное обеспечение использует классы (как основной элемент), а также наследование и инкапсуляцию, которые действуют на классы.

Объектно-ориентированное программное обеспечение является **событийно управляемым**. Передача управления внутри программы осуществляется не только путем явного указания последовательности обращений одних функций программы к другим, но и путем генерации сообщений различным объектам, разбора сообщений соответствующим обработчиком и передача их объектам, для которых данные сообщения предназначены.

При **процедурном программировании** контролируются потоки между функциями программного обеспечения.

Различие в подходах программирования требует дополнительных подходов при тестировании объектно-ориентированных приложений.

ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ ТЕРМИНОЛОГИЯ

КЛАСС - это план, согласно которому задаются данные и поведение (методы), общие для всех классов определенного типа.

ОБЪЕКТ - это представитель отдельного класса.

В качестве примера рассмотрим следующую аналогию.

> *Форма для печенья. Класс - это форма для печенья, которая задает то, как это печенье будет выглядеть. Объекты - это печенье, созданные с помощью какой-то одной формы.*

НАСЛЕДОВАНИЕ дает возможность получать новые классы из уже существующих классов. Дочерний класс (производный класс) наследует все данные и поведение (методы) коренного класса (базового класса). К нему также можно добавить новые функции или переопределить уже существующие.

ИНКАПСУЛЯЦИЯ обеспечивает программистов четко заданным интерфейсом для функций объекта таким образом, чтобы предотвратить возможность непосредственного доступа программиста к внутреннему коду объекта.

В объектно-ориентированном программировании имеется возможность ограничить доступ к членам класса (данным и методам). Объект всегда может получить доступ к членам, заданным в рамках его собственного класса. Члены класса, отмеченные как открытые, становятся внешне видимыми, позволяя объектам различных классов получать доступ к этим открытым членам. Закрытые члены, напротив, доступны только объектам данного класса, - даже производный класс не может получить доступ к частной информации родительского класса.

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

При тестировании программного обеспечения всегда задействуют несколько уровней, включая тестирование системы и поэлементное тестирование.

ТЕСТИРОВАНИЕ системы не зависит от процессов, используемых для создания какого-либо приложения. Тестер оценивает приложение с точки зрения пользователя. Поэтому для задания тестовых примеров на уровне системы знания об объектно-ориентированном программном обеспечении вовсе не обязательны.

При поэлементном тестировании любого приложения требуется знание о программировании и доступ к деталям реализации. В объектно-ориентированном коде класс – это наименьший элемент, который можно компилировать.

Главное отличие между двумя методологиями проектирования состоит в том, что при поэлементном тестировании процедурного программного обеспечения имеется непосредственный доступ к элементам (процедурам и функциям), в то время как доступ к данным и методам класса может быть более ограниченным. Поэлементное тестирование заключается в подтверждении того, что все методы в рамках класса и его интерфейс функционируют так, как было задумано.

Большинство методов задания тестов одинаково применимы как для процедурно разрабатываемых приложений, так и для объектно-ориентированных приложений. Основное отличие состоит в выполнении тестовых примеров. Поэлементное тестирование объектно-ориентированного приложения требует таких условий выполнения, при которых можно будет работать с инкапсуляцией.

При объектно-ориентированном программировании происходит переход от модели, описывающей структуру программы, к модели, описывающей поведение программы, что для тестирования можно классифицировать как положительное свойство данного перехода.

Отрицательным аспектом совершаемого перехода для применения рассмотренных ранее моделей является потеря заданных в явном виде связей между модулями программы.

Работа по тестированию приложения не должна включать в себя проверку работоспособности элементов библиотек, ставших фактически промышленным стандартом для разработки программного обеспечения, а только проверку кода, написанного непосредственно разработчиком программного проекта.

Тестирование объектно-ориентированной программы включает те же уровни, что и тестирование процедурной программы - модульное, интеграционное и системное.

Методика проведения тестирования программы, представленной в виде классовой модели программного проекта, включает в себя несколько этапов, соответствующих уровням тестирования.

- ✓ На первом уровне проводится тестирование методов каждого класса программы, что соответствует этапу модульного тестирования.

- ✓ На втором уровне тестируются методы класса, которые образуют контекст интеграционного тестирования каждого класса.
- ✓ На третьем уровне уже протестированный класс включается в общий контекст (дерево классов) программного проекта. Здесь становится возможным отслеживать реакцию программы на внешние события.

Второй и третий уровни рассматриваемой модели соответствуют этапу интеграционного тестирования.

В случае систем, написанных на процедурных языках, процесс тестирования модуля происходит так – для каждого модуля разрабатывается тестовый драйвер, вызывающий функции модуля и собирающий результаты их работы и набор заглушек, которые имитируют поведение функций, содержащихся в других модулях, не попадающих под тестирование данного модуля.

При тестировании объектно-ориентированных систем существует ряд особенностей, прежде всего вызванных инкапсуляцией данных и методов в классах.

Внутри класса отдельно взятые методы имеют императивный характер исполнения. Языки объектно-ориентированного программирования возвращают контроль вызывающему объекту, когда сообщение обработано. Поэтому каждый метод (функция - член класса) должен пройти традиционное модульное тестирование по выбранному критерию. Все результаты, полученные для тестирования модулей, безусловно, подходят для тестирования методов классов.

Процесс тестирования классов как модулей иногда называют компонентным тестированием. В ходе такого тестирование проверяется взаимодействие методов внутри класса и правильность доступа методов к внутренним данным класса. При таком тестировании возможно обнаружение не только стандартных дефектов, связанных с выходами за границы диапазона или неверно реализованными требованиями, а также обнаружение специфических дефектов объектно-ориентированного программного обеспечения:

- ✓ дефектов инкапсуляции, в результате которых, например, сокрытые данные класса оказывается недоступными при помощи соответствующих публичных методов;
- ✓ дефектов наследования, при наличии которых схема наследования блокирует важные данные или методы от классов-потомков;
- ✓ дефектов полиморфизма, при которых полиморфное поведение класса оказывается распространенным не на все возможные классы;
- ✓ дефектов инстанцирования, при которых во вновь создаваемых объектах класса не устанавливаются корректные значения по умолчанию параметров и внутренних данных класса.

Однако выбор класса в качестве тестируемого модуля имеет и ряд сопряженных проблем.

- ✓ **ОПРЕДЕЛЕНИЕ СТЕПЕНИ ПОЛНОТЫ ТЕСТИРОВАНИЯ КЛАССА**. Если в качестве тестируемого модуля выбран класс, не совсем ясно, как определять степень полноты его тестирования. С одной стороны, можно использовать классический критерий полноты покрытия программного кода тестами – если полностью выполнены все структурные элементы всех методов – как публичных, так и скрытых, то тесты можно считать полными. Однако существует другой подход к тестированию класса, согласно которому все публичные методы должны предоставлять пользователю данного класса согласованную схему работы и достаточно проверить типичные корректные и некорректные сценарии работы с данным классом.
- ✓ **ПРОТОКОЛИРОВАНИЕ СОСТОЯНИЙ ОБЪЕКТОВ И ИХ ИЗМЕНЕНИЙ**. Некоторые методы класса предназначены не для выдачи информации пользователю, а для изменения внутренних данных объекта класса. Значение внутренних данных объекта определяет его состояние в каждый отдельный момент времени, а вызов методов, изменяющих данные, изменяет и состояние объекта. При тестировании классов необходимо проверять, что класс адекватно реагирует на внешние

вызовы в любом из состояний. Однако, зачастую, из-за инкапсуляции данных невозможно определить внутреннее состояние класса программными способами внутри драйвера.

- ✓ **ТЕСТИРОВАНИЕ ИЗМЕНЕНИЙ**. Модульные тесты – мощный инструмент проверки корректности изменений, внесенных в исходный код при рефакторинге. Однако в результате рефакторинга только одного класса не меняется его внешний интерфейс с другими классами (интерфейсы меняются при рефакторинге сразу нескольких классов). В результате обычных изменений системы у класса может меняться внешний интерфейс, причем как по формальным признакам (изменяются имена и состав методов, их параметры), так и по функциональным (при сохранении внешнего интерфейса меняется логика работы методов). Для проведения модульного тестирования класса после таких изменений потребуется изменение драйвера и, возможно, заглушек. Но только модульного тестирования в данном случае недостаточно, необходимо также проводить и интеграционное тестирование данного класса вместе со всеми классами, которые связаны с ним по данным или по управлению.

Каждый класс должен быть рассмотрен и как субъект интеграционного тестирования. Интеграция для всех методов класса проводится с использованием инкрементальной стратегии снизу вверх. При этом мы можем использовать тесты для классов-родителей тестируемого класса, что следует из принципа наследования - от базовых классов, не имеющих родителей, к самым верхним уровням классов.

В ходе интеграционного тестирования должны быть проверены все возможные внешние вызовы методов класса, как непосредственные обращения, так и вызовы, инициированные получением сообщений.

Объектно-ориентированный подход, ставший в настоящее время неявным стандартом разработки программных комплексов, позволяет широко использовать иерархическую модель программного проекта. Каждый класс рассматривается как объект модульного и интеграционного тестирования. Сначала каждый метод класса тестируется, как модуль по выбранному критерию. Затем класс становится объектом интеграционного тестирования. Далее осуществляется интеграция всех методов всех классов в единую структуру - классовую модель проекта.

ЛИТЕРАТУРА

[Синицын 2006] – Синицын С.В., Налютин Н.Ю. Верификация программного обеспечения. Курс лекций. Московский инженерно-физический институт. М. 2006.

ПОЭЛЕМЕНТНОЕ ТЕСТИРОВАНИЕ КЛАССОВ

Классы являются характерной особенностью объектно-ориентированного программирования, однако подходы к заданию тестовых примеров часто остаются теми же, что и для процедурных приложений.

ТЕСТИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ОРТОГОНАЛЬНЫХ МАССИВОВ

У некоторых приложений число входных значений невелико и ограничено, но перечисление всех возможных перестановок порождает множество тестов. Одна из возможностей снизить количество тестовых примеров – это применение комбинаторных методов, известных как тестирование ортогональных массивов.

Тестирование ортогональных массивов основывается на статистических методах, заимствованных из промышленности. Для применения этого метода необходимо, чтобы у программы были независимые наборы состояний. Задача состоит в спаривании каждого состояния из одного набора со всеми состояниями из другого набора, по крайней мере, один раз.

Описание тестового примера по покупке книги пользователем.

Приложение для книжного магазина обрабатывает информацию представленную на рис.6.11 [Тампе 2006].

Книга	Закупка	Доставка
в_наличии_на_складе	наличные	срочная
специальный_заказ	чек	экономная
распродана	долговое обязательство	по месту
		задержка

Рис. 6.11. Классы и состояния в примере с книжным магазином

Программное обеспечение, созданное с помощью объектно-ориентированных методов, состоит из трех классов: книга, покупка и доставка, которые имеют ограниченное число возможных состояний.

В процедурных приложениях три процедуры – заказать_книгу, купить_книгу и доставить_книгу – обладают аргументами с конечным набором значений. Для упрощения будем использовать термины класс и состояние.

У каждого из трех классов есть три состояния, а у одного класса их четыре, так что тестирование каждой комбинации потребует 32×41, или 36 тестовых примеров. Если выбрать состояние из двух классов (известное как двухточечная комбинация), то потребуется только 12 тестовых примеров, показанных в табл.6.4 [Тампе 2006], и, следовательно, число необходимых тестовых примеров будет ниже.

Таблица 6.4. Применение ортогональных массивов

Тестовый пример	Книга	Закупка	Доставка
1	в_наличии_на_складе	наличные	срочная
2	в_наличии_на_складе	чек	экономная
3	в_наличии_на_складе	долговое обязательство	по месту
4	в_наличии_на_складе	наличные	задержка
5	специальный_заказ	чек	срочная
6	специальный_заказ	долговое обязательство	экономная
7	специальный_заказ	наличные	по месту
8	специальный_заказ	чек	задержка
9	распродана	долговое обязательство	срочная
10	распродана	наличные	экономная
11	распродана	чек	по месту
12	распродана	долговое обязательство	задержка

Рассматривая в таблице только столбцы “Книги” и “Доставка” (игнорируя столбец “Закупка”), можно увидеть всевозможные комбинации этих двух классов. Аналогично, игнорируя класс книга, прослеживаются всевозможные комбинации состояний закупки и доставки. У класса доставка больше состояний, чем у остальных двух классов, поэтому инспектирование спариваний столбцов “Книги” и “Закупка” указывает на то, что все возможные комбинации дублируются. Один из случаев дублирования представлен тестовыми примерами 1 и 4, где классы книга и закупка обладают соответственно состояниями в_наличии_на_складе и наличные. Эти два теста по-прежнему остаются уникальными по причине различных состояний доставки.

Способ реализации тестового примера (например, №1) зависит от среды разработки. При объектно-ориентированном программировании задача данного теста заключается в том, чтобы объект класса книга в состоянии в_наличии_на_складе отправил сообщение, которое проходит объект класса закупка в состоянии наличные к объекту класса доставка в состоянии срочная. В процедурном языке этот тест заключается в передаче параметров в_наличии_на_складе, наличные и срочная процедурам заказать_книгу, купить_книгу и доставить_книгу соответственно.

Тестирование ортогонального массива – это мощный метод, который используется всякий раз, когда у системы прослеживается небольшая область входных данных и большое число перестановок.

ТЕСТИРОВАНИЕ НАСЛЕДИЯ

При наследовании создаются новые классы, которые повторно используют и расширяют классы, созданные ранее. При этом допускается расширение функций существующего кода без реальной модификации реального кода. Производный класс, или потомок родительского класса, наследует те же возможности, которые есть и у исходного родительского класса, хотя при этом также добавляются его собственные возможности.

Рассмотрим характерный пример. На рис.6.12 [Тампе 2006] показана структура наследования для трех классов: 1) Класс А; 2) Класс Б, потомок класса А; 3) Класс В, потомок класса Б и класса А.

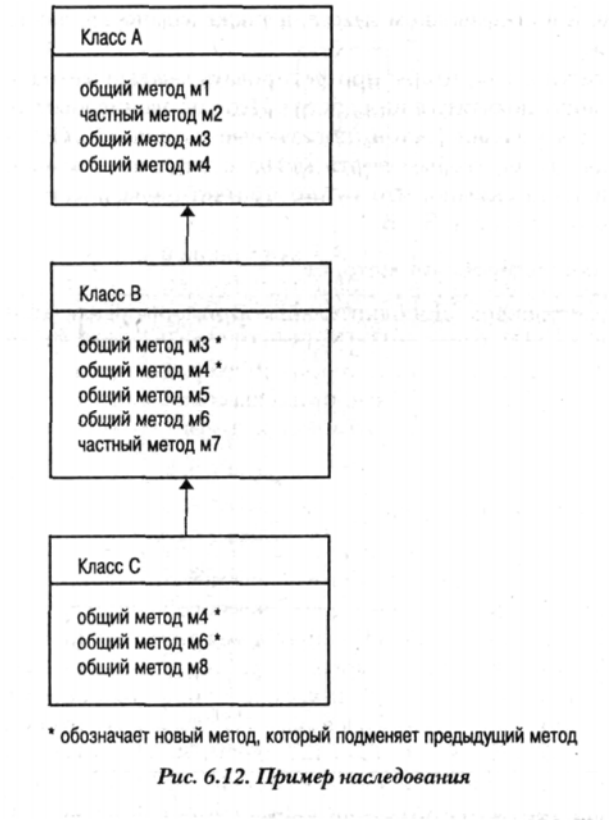

Рис. 6.12. Пример наследования

Даже, если предыдущее тестирование подтвердило, что родительский класс сам по себе работает корректно, остается вопрос, будут ли работать методы, заданные в родительском классе, если их вызывает дочерний.

Проектирование с неглубокой структурой наследования проще для тестирования. При выравнивании классов создается диаграмма, которая выделяет все унаследованные функции. В диаграмме, показанной на рис.6.13 [Тампе 2006], перечислены все методы, как в родительском, так и производном классе, а также задано то, какие методы видны для каждого класса.

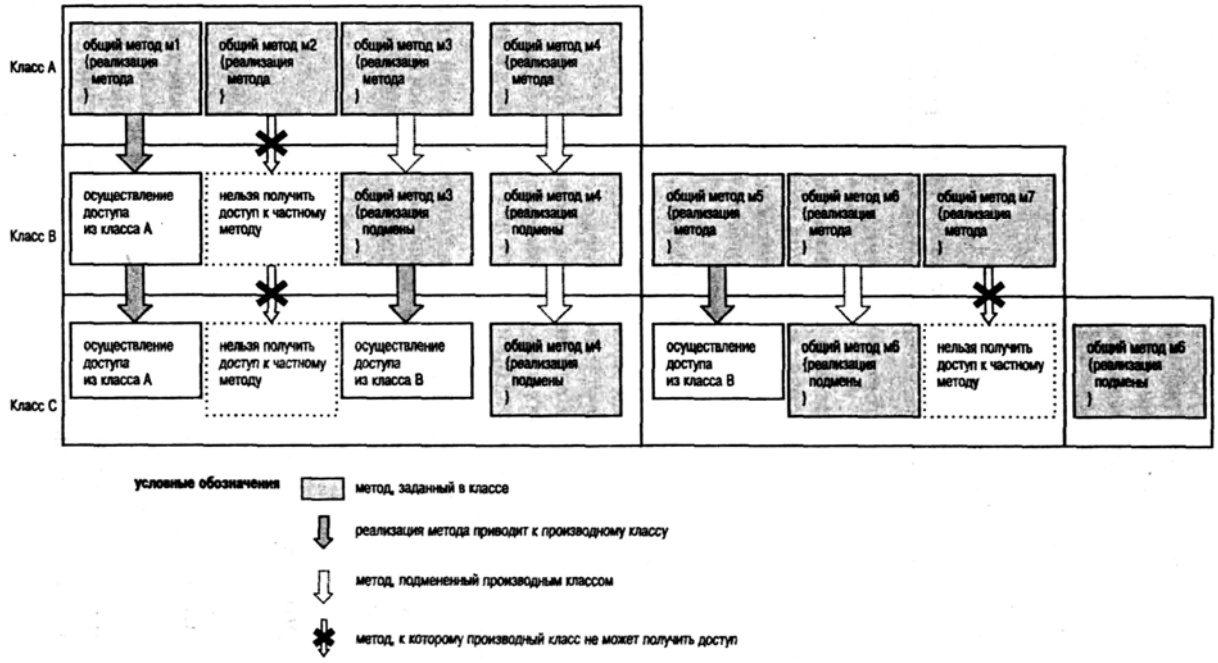

Рис. 6.13. Выравнивание иерархии классов

Задача заключается в том, чтобы протестировать каждый метод в контексте всех классов, которые могут получить к ним доступ. Использование диаграммы на рис.6.13 помогает определить условия тестов, показанные в табл.6.5 [Тампе 2006].

Таблица 6.5. Условия тестирования методов

Метод	Тест из этого класса
А.м1	класс А, Б, В
А.м2	класс А
А.м3	класс А
Б.м4	класс Б, В
А.м4	класс А
Б.м4	класс Б
В.м4	класс В
Б.м5	класс Б, В
Б.м6	класс Б
В.м6	класс В
Б.м7	класс Б
В.м8	класс В

Обозначение “класс, метод” принято для того, чтобы связать метод с классом, в котором он задается. В первом условии теста сказано, что нужно протестировать метод м1 (заданный в классе А) в контексте классов А, Б и В.

СЛОЖНОСТИ ПРОВЕДЕНИЯ ТЕСТОВ

Следующий этап разработки тестовых примеров – разработка среды, в которой будут проводиться тесты. Для этого требуется специальное тестовое программное обеспечение (драйвер), которое будет активизировать тестируемое программное обеспечение. Типичный драйвер выполняет такие функции:

- ✓ упаковки входных и других данных;
- ✓ активизации тестируемого программного обеспечения;
- ✓ регистрации исходных результатов.

Создание драйвера для тестируемого класса требует специального обсуждения. Драйверы должны обходить инкапсуляцию, чтобы иметь возможность контролировать и наблюдать за внутренними данными и методами класса.

Существует несколько способов, которыми можно этого достичь:

- ✓ для каждого класса предусматривается метод тестового примера;
- ✓ создаются параллельные, идентичные исходным классы, которые отличаются наличием дополнительного кода необходимого для тестирования;
- ✓ создаются дочерние классы, которые наследуют тестируемые методы.

Все эти подходы видоизменяют классы и, следовательно, модифицируют реальное приложение. Таким образом, тестируемая реализация может оказаться неидентичной выпускаемому коду. Важно, чтобы среда тестирования была как можно ближе к среде разработки, в противном случае нельзя будет полностью протестировать конечный продукт.

ЛИТЕРАТУРА

[Тампе 2003] – Тампе Л. Введение в тестирование программного обеспечения.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2003.

ПРИМЕР ТЕСТИРОВАНИЯ СИСТЕМЫ

ОПИСАНИЕ ТЕСТОВОГО ПРИМЕРА

В системе “Путеводитель” для местного торгового центра есть несколько торговых точек, которые относятся к следующим категориям.

- ✓ Женская одежда (3 торговые точки).
- ✓ Мужская одежда (2 торговые точки).
- ✓ Детская одежда (4 торговые точки).
- ✓ Женская обувь (2 торговые точки).
- ✓ Мужская обувь (2 торговые точки).
- ✓ Цветы (1 торговая точка).
- ✓ Подарки (5 торговых точек).
- ✓ Рестораны (5 торговых точек).

Путеводитель по пассажу взаимодействует с двумя типами действующих лиц:

- ✓ покупатели, которые разыскивают нужную торговую точку в пассаже;
- ✓ администраторы, которые обновляют названия торговых точек и их размещение.

Входные данные пользователя описаны на рис.6.1 [Тампе 2006].

Команды покупателя: главное меню

1. Покупатель прикасается к клавише названия одной из категорий торговых точек в списке и получает список торговых точек, из которых можно выбирать.
2. Покупатель прикасается к клавише карты и на экране появляется карта пассажа.
3. Покупатель прикасается к клавише справки и получает экран справки.

Команды покупателя: список торговых точек

4. Покупатель прикасается к клавише названия конкретной торговой точки и на экране появляется карта, на которой высвечивается положение этой торговой точки.
5. Покупатель прикасается к клавише главного меню и возвращается обратно к главному меню.
6. Покупатель прикасается к клавише справки и получает экран справки.

Команды покупателя: карты

7. Покупатель прикасается к кнопке печати и получает распечатку нужной карты.
8. Покупатель прикасается к клавише ВЕРНУТЬСЯ НАЗАД и получает отображение предыдущего экрана.
9. Покупатель прикасается к клавише главного меню и возвращается к главному меню.
10. Покупатель прикасается к клавише справки и переходит к справочному меню.

Команды покупателя: справка

11. Покупатель нажимает клавишу ВЕРНУТЬСЯ НАЗАД и получает отображение предыдущего экрана.
12. Покупатель прикасается к клавише главного меню и возвращается к главному меню.

Команды администратора

13. Администратор добавляет название новой торговой точки.
14. Администратор удаляет название торговой точки.
15. Администратор видоизменяет карту пассажа.

Рис. 6.1. Варианты входных данных для путеводителя по пассажу

На экране главного меню (рис.6.2 [Тампе 2006]) у пользователя запрашивается тип торговой точки.

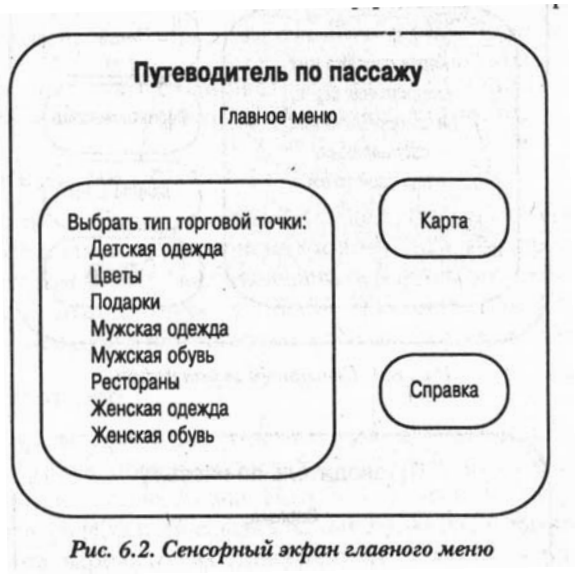

Рис. 6.2. Сенсорный экран главного меню

На рис.6.3 [Тампе 2006] представлено меню со списком торговых точек.

Рис. 6.3. Сенсорный экран списка торговых точек

Экран с главным меню и экран со списком торговых точек позволяет покупателю сделать запрос на печатную карту пассажа (рис.6.4 [Тампе 2006]).

Рис. 6.4. Сенсорный экран карты

На всех экранах у пользователя имеется возможность выбрать функцию справки (рис.6.5 [Тампе 2006]).

Рис. 6.5. Сенсорный экран справки

Текущие требования не предусматривают каких-либо подробностей в отношении интерфейса администратора.

Входная пользовательская информация, представленная в этом разделе, играет роль требований к “Путеводителю”. Для разработки тестовых примеров на уровне системы существуют различные стратегии.

В ПЕРВОЙ СТРАТЕГИИ используется схематический подход. Схема служит полезным инструментом для группировки взаимосвязанной информации.

ВО ВТОРОЙ СТРАТЕГИИ используются прецеденты, которые ориентированы на сценарии действующих лиц.

Эти независимые примеры приводят в результате к похожим, хотя и неидентичным тестовым примерам. На выбор тестером определенного подхода влияет несколько факторов:

- ✓ опыт и личные предпочтения тестера;
- ✓ формат, в котором представлены системные требования;
- ✓ наличие инструментов, которые могут помочь в разработке тестовых примеров.

Иногда тестер может начать с одной стратегии, а затем переключиться на другую, если с использованием исходного метода возникают трудности.

ТЕСТОВЫЕ ПРИМЕРЫ НА ОСНОВЕ СХЕМЫ

Поскольку описание входных данных имеет вид списка, организация этой информации в схему является вполне обоснованным подходом. В этом примере, чтобы не усложнять описание тестовых примеров, интерфейсы покупателя и администратора будут обрабатываться отдельно.

КОМАНДЫ ПОЛЬЗОВАТЕЛЯ

Схему можно создать, проводя реорганизацию вариантов входных данных в последовательность, которую обычно задает пользователь. Список категорий торговых точек также становится частью схемы. На рис.6.6 [Тампе 2006] показана результирующая схема. В квадратных скобках содержится ссылка на исходное требование. Скобки с числами показывают на описание вариантов входных данных, показанное на рис.6.1, в то время как элементы, помеченные [Д], ссылаются на текущую конфигурацию пассажа.

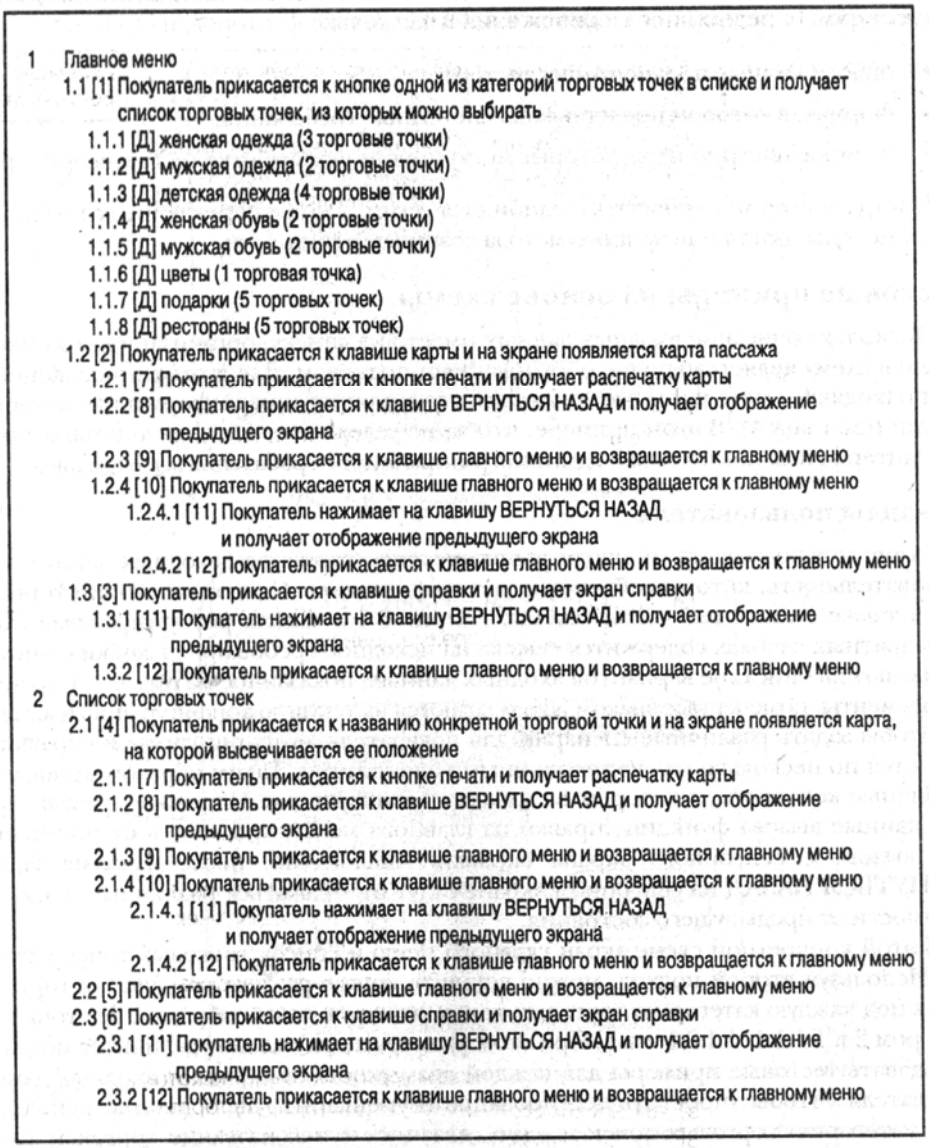

Рис. 6.6. Схема для входной последовательности покупателя

Чтобы задать различные сценарии для покупателя, функции карты и справки появляются по несколько раз под различными элементами. Таким образом, освещаются различные контексты, в которых вызываются эти функции.

В этой схеме экран главного меню и список торговых точек разделены. Используя второй подход, можно вставить копию раздела со списком торговых точек под каждую категорию торговых точек, копируя, таким образом, все точки под номерами 2 в 1.1.1.1, 1.1.2.1. и т.д. При этом размер схемы сильно увеличится и придется задавать тестовые примеры для каждой возможной комбинации входных данных покупателя. Чтобы упростить тестирование и уменьшить число тестов, используем одну категорию торговых точек и одно связанное с ней название торговой точки, чтобы можно было выполнить сценарий под элементом №2.

Представленная схема – лишь одна из возможных интерпретаций описания пользовательского интерфейса. Разные тестеры создают различные схемы, но суть остается той же. Основная задача заключается в перечислении действий пользователя, чтобы можно было создать тесты, охватывающие все наиболее важные сценарии.

Схема состоит и 23 листьев, каждый из которых дает в результате тестовый пример, некоторые из них показаны в табл.6.1 [Тампе 2006].

Таблица 6.1. Выбор тестовых примеров на уровне системы для интерфейса покупателя (полный список содержится в приложении В1)

ID тестового примера	Источник теста	Входные данные	Ожидаемый результат
ПП-1	1.1.1	Из главного меню: выбрать "Женская одежда"	Появляется экран со списком торговых точек и перечисляются 3 торговые точки с женской одеждой
.			
.			
.			
ПП-5	1.1.5	Из главного меню: выбрать "Мужская обувь"	Появляется экран со списком торговых точек и перечисляется 2 торговые точки с мужской обувью
.			
.			
.			
ПП-11	1.2.3	Из главного меню: а) нажать кнопку КАРТА б) нажать кнопку ГЛАВНОЕ МЕНЮ	а) Выдается карта пассажа, размещенная на экране б) Возврат к главному меню
.			
.			
.			
ПП-17	2.1.2	Из главного меню: а) выбрать "Мужская одежда" б) выбрать название второй торговой точки из списка в) нажать кнопку ВЕРНУТЬСЯ НАЗАД	а) Список торговых точек и список из 2 торговых точек мужской одежды б) Выдается карта пассажа, размещенная на экране в) Возврат к экрану со списком торговых точек, на котором перечислены две торговые точки мужской одежды
.			
.			
.			

КОМАНДЫ АДМИНИСТРАТОРА

Для создания тестов необходимо понять, как администратор взаимодействует с системой (см. схему на рис.6.7 [Тампе 2006]). Большая часть добавленных элементов была получена на основе разумных предположений, которые охватывают множество возможных ситуаций. Объединение различных условий приводит к всесторонности тестовых примеров.

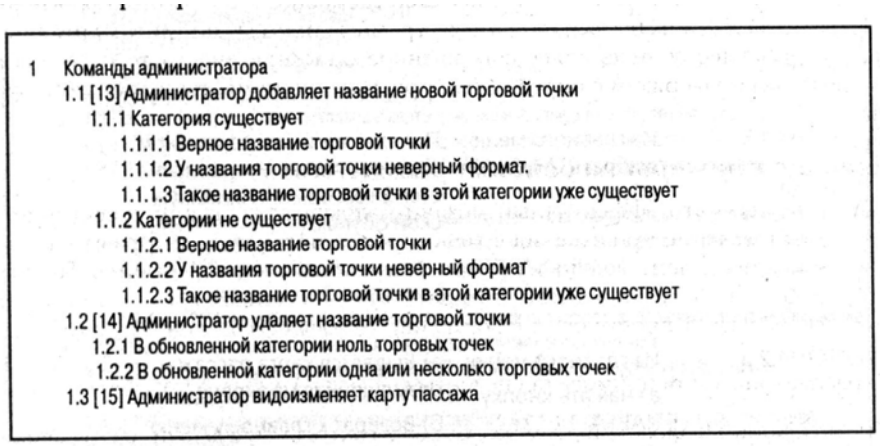

Рис. 6.7. Исходная схема для интерфейса администратора

Когда в требованиях отсутствует нужная информация, тестеру лучше всего обратиться с этой проблемой к авторитетному лицу проекта. Ниже перечислена информация, которой не хватает в рассматриваемом примере.

1. Как администратор выбирает категории, чтобы добавить торговую точку в существующую категорию.
2. Как администратор создает новую категорию. Что если название категории уже существует.
3. Каков формат названий категорий.
4. Можно ли присвоить одно название нескольким категориям.
5. Когда администратор выбирает название перемещаемой торговой точки, должен ли он также задавать категорию.
6. Что произойдет при перемещении названия торговой точки, если во время обновления категория окажется пустой.
7. Как администратор задает название торговой точки, которое нужно переместить (указывает и щелкает на изображение карты; выбирает название торговой точки из списка; набирает название торговой точки).
8. Как администратор задает название торговой точки, которую нужно добавить.
9. Каким образом администратор задает новое расположение торговой точки. Набирает номер, соответствующий расположению на карте.
10. Может ли администратор изменить расположение существующей торговой точки, щелкая и перетягивая изображение на карте пассажа.

После того как авторитетное лицо проекта доопределит требования, тестеры заканчивают составление схемы, а затем переходят к заданию тестовых примеров.

ТЕСТОВЫЕ ПРИМЕРЫ НА ОСНОВЕ ПРЕЦЕДЕНТОВ

ПРЕЦЕДЕНТЫ – это эффективный механизм документирования требований. Они описывают поведение приложения с точки зрения пользователя. Правильно сконструированный прецедент можно легко преобразовать в тестовый пример.

В рассматриваемом примере прецеденты состоят из четырех разделов.

1. **ОСНОВНОЙ ЗАГОЛОВОК**. В этом разделе содержится уникальный идентификатор и краткое описание прецедента.
2. **ГАРАНТИИ УСПЕХА**. В этом разделе описывается конечное состояние, возникающее при успешном завершении теста.

3. **ОСНОВНОЙ УСПЕШНЫЙ СЦЕНАРИЙ**. В разделе описывается простейший сценарий, в котором каждый этап завершается нормально и нет необходимости в восстановлении системы.
4. **РАСШИРЕНИЕ**. Здесь перечисляются альтернативные возможности, которые ответвляются от этапов в основном успешном сценарии. В зависимости от результирующих действий расширения могут возвращаться к основному сценарию или достигать своего собственного завершения (например, генерируя неудачу или совершенно отличный результат).

В прецедентах содержится дополнительная информация (например, предусловия, основные действующие лица, неудачные конечные условия, триггеры). Три таких прецедента показаны на рис.6.8-6.10 [Тампе 2006].

Прецедент № 1. Покупатель ищет в пассаже определенную торговую точку.

Гарантии успеха

Покупатель получает распечатку карты для желаемой торговой точки.

Основной сценарий успеха

1. Покупатель выбирает категорию торговых точек.
2. Покупатель выбирает название торговой точки.
3. Покупатель выбирает функцию печати.

Дополнения

1а. Покупатель хочет получить справку при выборе категории торговой точки.
1б. Нет подходящей категории торговых точек.
2а. Покупатель хочет получить справку при выборе названия торговой точки.
2б. Желаемой торговой точки не существует.

Рис. 6.8. Прецедент № 1

Прецедент № 2. Покупатель ищет все торговые точки с подарками и хочет получить карту с кратчайшим путем к каждой из них.

Гарантии успеха

Покупатель получает распечатку карты для каждой торговой точки с подарками.

Основной сценарий успеха

1. Покупатель выбирает категорию торговой точки.
2. Покупатель выбирает название первой торговой точки.
3. Покупатель выбирает функцию печати.
4. Покупатель возвращается к списку торговых точек.
5. Покупатель выбирает второе название торговой точки.
6. Покупатель выбирает функцию печати.

Дополнения

(нет)

Рис. 6.9. Прецедент № 2

Прецедент № 3. Администратор добавляет в "Путеводитель" новую торговую точку.

Гарантии успеха

В списке торговых точек появляется новое название.

На карте пассажа появляется новое положение.

Основной сценарий успеха

1. Администратор выбирает категорию торговой точки.
2. Администратор добавляет название торговой точки.
3. Администратор обновляет карту.

Дополнения

1а. Нет подходящей категории.

2а. Название торговой точки уже существует.

3а. Выбранное положение уже выделено под другую торговую точку.

Рис. 6.10. Прецедент № 3

Каждый прецедент позволяет сгенерировать несколько тестовых примеров. В основном успешном сценарии, который является самым простым и наиболее общим описанием прецедента, находится информация для первого прецедента. Данный сценарий также служит основой для последующих тестов. Чтобы убедиться, что система правильно оперирует этим событием, для каждого расширения должен существовать, по крайней мере, один тестовый пример. В табл.6.2 [Тампе 2006] перечислены тестовые примеры, полученные на основе прецедента №1.

Таблица 6.2. Тестовые примеры для прецедента № 1

ID тестового примера	Источник тестового примера	Исходное состояние системы	Входные данные	Ожидаемые результаты	Реальные результаты
п1-1	прецедент № 1	Начать тест с главного меню	а) выбрать "Мужская одежда" б) выбрать название первой торговой точки в) нажать кнопку печати	а) Появляется список экранов с торговыми точками и перечисляется две торговые точки мужской одежды б) Выдается карта с этой торговой точкой, размещенная на экране (в) Выдается печатная копия карты	
п1-2	прецедент № 1 расширение 1а	Начать тест с главного меню	а) нажать кнопку СПРАВКА б) нажать кнопку ВЕРНУТЬСЯ НАЗАД	а) Появляется экран со справкой, предоставляющий информацию относительно главного меню б) Возврат к главному меню	
п1-3	прецедент № 1 расширение 1б	Начать тест с главного меню	[Нет входных данных: Покупатель ищет книжную торговую точку]	В главном меню отсутствует категория книжных торговых точек	
п1-4	прецедент № 1 расширение 2а	Начать тест с главного меню	а) Выбрать пункт "Рестораны" б) нажать кнопку СПРАВКА	а) Появляется экран со списком торговых точек, на котором перечислено 5 ресторанов б) Выдается экран со справкой относительно меню торговых точек	
п1-5	прецедент № 1 расширение 2б	"Обувь Лорны" – это неверное название торговой точки Начать тест с главного меню	Выбрать пункт "Женская обувь"	Появляется экран со списком торговых точек, на котором перечисляются две торговые точки женской обуви, где нет магазина "Обувь Лорны"	

Этот набор тестовых примеров явно отличается от полученного в разделе 6.3.1 – в данном примере прецеденты описывают намерения действующих лиц, тогда как в схеме перечисляются комбинации

входных данных. Здесь нет требований, в которых указывалось бы, что произойдет, если покупателю не удастся найти желаемую торговую точку в пассаже, как в тестовом примере п1-3.

Разные тестеры могут предлагать разные тестовые примеры, которые являются производными одного и того же прецедента. В матрице перекрестных ссылок в табл.6.3 [Тампе 2006] отслеживается, какие из вариантов входных данных (из списка на рис.6.1) соответствуют тестовому примеру.

Таблица 6.3. Отображение входных данных пользователя на тестовые примеры

Вариант входных данных	п1-1	п1-2	п1-3	п1-4	п1-5	п2-1	п3-1	п3-2	п3-3	п3-4
1	✓			✓	✓	✓	✓	✓		
2										
3		✓								
4	✓					✓	✓	✓		
5										
6				✓						
7	✓					✓	✓			
8						✓				
9										
10										
11		✓								
12										
13							✓	✓	✓	✓
14										
15							✓	✓		✓

Матрица не может адекватно отображать команды администратора вследствие неполноты требований. Даже если в матрице показано, что в четырех тестовых примерах используется вариант входных данных №13 (добавить название новой торговой точки), не существует различия между успешным добавлением, отсутствием категории торговых точек или неверным названием торговой точки, хотя все они являются отдельными ситуациями, которые относятся к разным прецедентам. Несмотря на эти недостатки, матрица помогает обнаружить необходимость в дополнительных тестовых примерах.

Не только в объектно-ориентированном программном обеспечении, но и во многих приложениях прецеденты используются для полного документирования системных требований. Прецеденты помогают преобразовывать описание пользовательского интерфейса в тестовые примеры.

ЛИТЕРАТУРА

[Тампе 2003] – Тампе Л. Введение в тестирование программного обеспечения.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2003.