Эти плановые документы суть рабочие документы. После каждого приращения, а иногда и после специальных итераций, эти документы пересматриваются и подвергаются ревизии. Корректируются степени рисков, равно как приоритеты и временные графики.

Планирование работ по тестированию

Планы тестирования классов передаются на усмотрение разработчика по мере того, как тестирование становится целесообразным или необходимым. Тестирование классов имеет смысл выполнять в процессе написания программных кодов, когда разработчик желает знать, какие детали упущены, или убедиться в правильности некоторой части реализации. Тестирование классов становится необходимым, когда некоторый компонент нужно добавить к базовому программному коду. Полностью разработка класса может быть не завершена, но поведение, которое он представляет, должно быть правильным и должно быть представлено в завершенном виде.

Комплексные испытания обычно планируется проводить через заданные интервалы времени, обычно в конце крупных итераций, которые означают завершение очередного приращения, и/или перед выпуском очередной версии программного продукта. С другой стороны, интеграция может быть продолжающимся процессом, с высокой степенью итераций, который повторяется в конце каждого рабочего дня. Циклы комплексных испытаний могут быть спланированы так, чтобы они совпадали с поставками крупных сторонних производителей программного обеспечения, например, новых версий базовых структур.

Системные тесты будут выполняться на крупных комплектующих модулях в заданные интервалы времени на протяжении всего цикла разработки проекта. Этот график обычно описан в плане выполнения проекта, поскольку часто требуется координация усилий с отдельными организациями, которые, возможно, оказывают свои услуги одновременно большому числу проектов.

Оценка

Оценка ресурсов, т.е. денежных средств, времени и персонала, которые потребуются для поддержки разрабатываемых планов, является частью этих планов. Это нелегко сделать, и у нас для этого нет «магической» формулы. В этом разделе обсуждаются такие факторы, как уровень покрытия, тип предметной области, необходимое оборудование, организационная модель и затраты труда на тестирование, которые учитываются при планировании.

Уровень покрытия

Чем выше необходимый уровень покрытия, тем больше потребуется для этого ресурсов. В литературе даются разные оценки количества программных кодов, необходимых для поддержки тестирования. Бейцер в [Beyz90] оценивает их количество на уровне от 2% до 80% от размера приложения. Нам удалось получить сравнительно точную оценку после того, как мы приняли каждый случай

использования в качестве единицы измерения. Получив оценку трудозатрат на каждый случай использования (возможно, через создание прототипа), можно определить трудозатраты на всю систему. Некоторые случаи использования отличаются от остальных большей областью видимости и большим уровнем абстракции. Выберите случаи использования, которые характеризуются примерно одинаковым уровнем детализации в моделях, и используйте их для оценки. Если два каких-нибудь случая использования входят в некоторый обобщенный случай использования, расширяя его, то следует воспользоваться либо двумя более специальными случаями использования, либо одним обобщенным, но никак не обоими типами случаев.

Тип предметной области

Довольно часто программное обеспечение, ориентированное на сложную обработку данных, переносит эту сложность в программную логику, при этом ввод данных достаточно прост. С другой стороны, системы, осуществляющие обработку крупных массивов данных, не отличаются особой сложностью программной логики, однако построение тестовых примеров потребует значительных усилий. Величина трудозатрат для построения завершенного тестового примера, включая полные наборы входных данных и правильные наборы выходных данных, меняется в широких пределах. Простая программа, которая обращается с запросами к крупной базе данных, требует много времени для построения соответствующих наборов данных и много времени для проверки правильности ответа.

Требуемые технические средства

Системное тестирование должно проводиться в среде, максимально близкой к рабочей среде. Даже некоторые аспекты тестирования классов могут потребовать специальной аппаратной среды или моделирования аппаратных средств. Затраты на установку и обслуживание аппаратных средств или построение имитатора должны быть включены во все сметы и калькуляции.

Организационная модель

Мы обсудили две схемы, которые широко используются для комплектования процессов тестирования. Наш опыт показывает, что чем большей независимостью пользуются тестировщики из организации, осуществляющей разработку, тем выше качество тестов. В то же время, подобная независимость требует более продолжительной кривой обучения и, следовательно, увеличения затрат. По общему признанию, один независимый тестировщик может справиться с результатами труда не более чем двух-трех разработчиков.

И наоборот, сильная привязка тестировщиков к организации, выполняющей разработку (или привлечение специалистов из коллектива разработчиков к тестированию), уменьшает затраты времени на ознакомление с системой. Спецификации редко когда представлены в завершенном виде или соответствуют пос-

ледним требованиям. Если тестировщиком будет лицо, принимающее участие в обсуждении решений, то такой тестировщик лучше понимает скрытый смысл требований спецификации. В то же время таким тестировщикам труднее отстаивать собственное мнение и сохранять объективность, если они вовлечены в разработку тестируемого программного продукта.

Рассмотрите возможность применения **дружественного подхода** к тестированию классов. Он обеспечивает высокую объективность оценок, благодаря чему тестирование становится более эффективным. Вместо того чтобы заставлять разработчиков тестировать свои собственные классы, сформируйте дружественные группы разработчиков. Два разработчика обмениваются между собой программными кодами и выполняют тестирование. Поскольку тестировщик является одновременно и разработчиком, который разрабатывает близкие по содержанию программы, он быстро входит в курс дела и приступает к тестированию гораздо быстрее, чем член независимого коллектива тестировщиков, который сначала должен тщательно изучить контекст.

Оценка трудозатрат на тестирование

Методы тестирования почти всегда опираются на предысторию, что позволяет выступать с прогнозами. Мы не будем растрачивать пространство данной книги на обсуждение этих методов. На рис. 3.12 представлена очень простая форма, предназначенная для отчета за каждый час времени, затраченного на выполнение различных видов тестирования. По мере изучения материала данной книги мы будем давать все более подробные инструкции по заполнению различных граф этой формы.

На данный момент можно подвести итог по большинству пунктов этой формы, воспользовавшись предысторией с целью определения стоимости разработки отдельных классов. Из списка, представленного на рис. 3.12, видно, какие классы требуется построить:

- Для каждого класса приложения, который нужно тестировать в автономном режиме, построить класс PACT.[6]
- Для каждого случая использования построить класс PAST.[7]
- Определить, какое количество классов необходимо для построения рассматриваемой инфраструктуры.

Общее количество классов, помноженное на трудозатраты, дает суммарные трудозатраты на тестирование классов. Вопросы планирования рассматриваются в разделе «Затраты на планирование».

---

[6] PACT (Parallel Architecture for Component Testing) — это параллельная архитектура для тестирования компонентов. Она будет рассматриваться в главе 7.

[7] PAST (Parallel Architecture for System Testing) — это параллельная архитектура для тестирования системы. Она будет рассматриваться в главе 9.

Форма подсчета трудозатрат на тестирование

Количество человеко-часов, необходимых для составления плана тестирования класса на уровне разработчиков	_____
Количество человеко-часов, необходимых для построения плана тестирования класса PACT на уровне разработчиков	_____
Число человеко-часов, необходимых для построения силами штатных работников среды тестирования	_____
Количество человеко-часов, необходимых для планирования силами тестировщиков системных испытаний	_____
Количество человеко-часов, необходимых для построения силами тестировщиков класса PAST.	_____
Общие трудозатраты тестировщиков в человеко-часах	_____

Рисунок 3.12. *Форма подсчета трудозатрат на тестирование*

Процесс тестирования игры «Кирпичики»

В этом разделе будет показано, как работают описанные выше пять оцениваемых факторов применительно к исследуемым случаям использования:

- *Кто выполняет тестирование?* Обязанности по тестированию будут распределены между двумя авторами этой книги. Сайкс (Sykes) выполняет основную часть реализации, и в силу этого обстоятельства он будет осуществлять тестирование классов и комплексные испытания. Макгрегор (McGregor) написал случаи использования и выполнил большую часть проектирования на верхнем уровне. Он будет создавать тестовые примеры на базе случаев использования и выполнять их прогон, как только будет готова реализация системы. Сайкс будет координировать тестирование моделей.
- *Какие компоненты программного обеспечения подвергнутся тестированию?* Тестироваться будут основные базовые классы. Классы высокого уровня, которые построены на базе примитивных классов, обросли таким большим числом взаимодействий, что при тестировании они будут рассматриваться как кластер. Система в ее окончательном виде будет тестироваться как завершенное приложение.
- *В какой момент следует выполнять тестирование?* Тестирование классов будет многократно производиться на протяжении циклов разработки. Групповое тестирование классов высокого уровня также будет повторяться на протяжении циклов разработки, но эти испытания не могут быть начаты до начала работ со вторым комплектующим модулем, т.е. пока не будут завершены примитивные классы в первом комплектующем модуле. Системное тестирование начнется сразу же после того, как станет доступной исходная версия системы по завершении работ над первым комплектующим модулем.

- *Как будет выполняться тестирование?* Тестовые примеры будут построены в виде методов класса. Для каждого класса программного продукта будет построен класс тестирования. Тестирование случаев использования будет выполняться специалистом по системам, и никакие виды автоматизации при этом применяться не будут. Это требует многократного выполнения сеансов игры.
- *Какой объем тестирования можно считать адекватным для конкретного компонента?* Классы будут подвергаться тестированию до тех пор, пока каждый общедоступный метод не будет вызван хотя бы раз. Мы не будем стремиться проверить каждую возможную комбинацию значений параметров. Тестовые примеры, берущие свое начало от случаев использования, не могут покрыть всех результатов.

Шаблоны документов

Мы обсудим план тестирования проекта, план тестирования компонентов, план комплексных испытаний, план тестирования случаев использования и план системных испытаний. Рисунок 3.13 служит иллюстрацией взаимозависимостей перечисленных планов. Каждая стрелка на рисунке означает, что документ, на который она указывает, включен по ссылке в документ, от которого исходит эта стрелка.

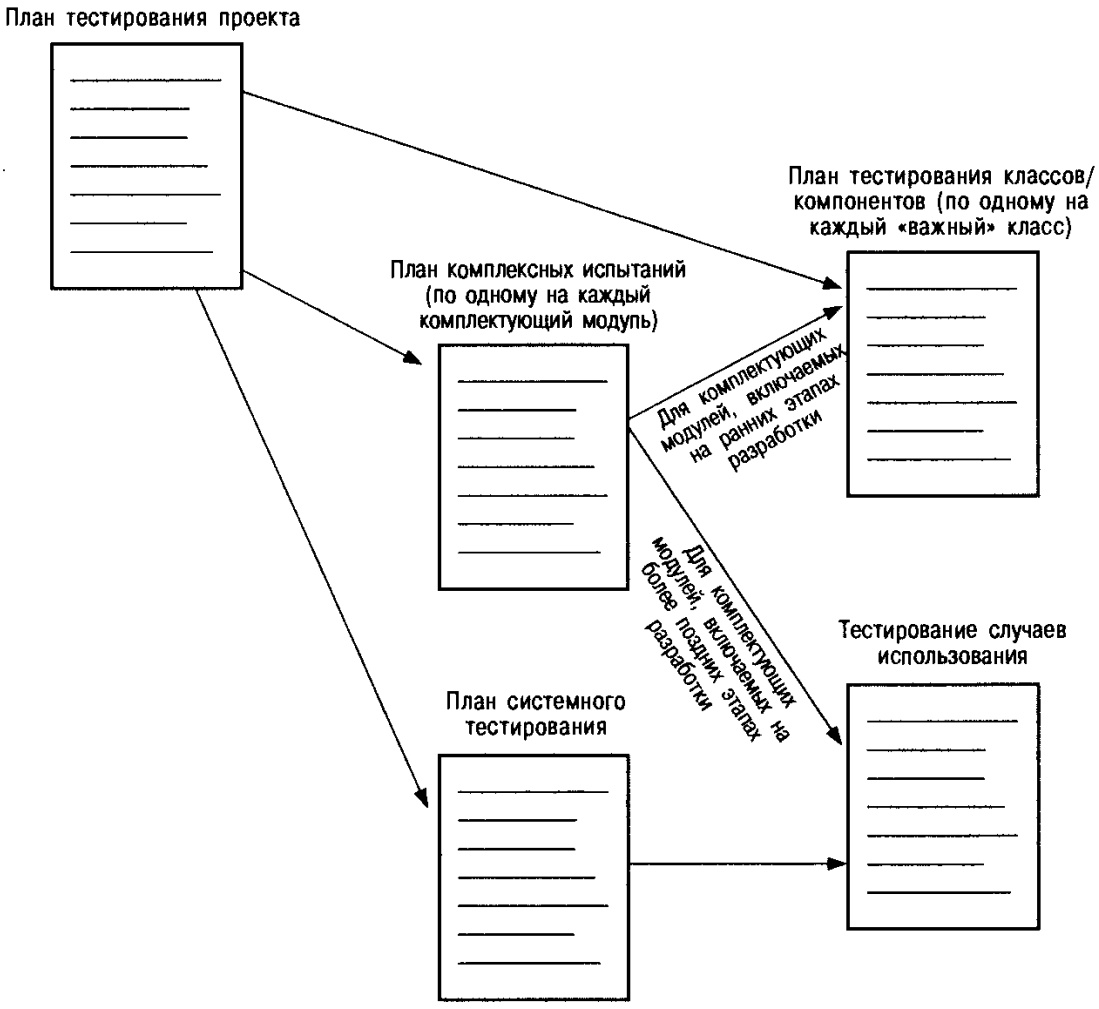

Рисунок 3.13. *Взаимозависимость планов тестирования*

Мы представим это в формате шаблона. Такой подход полезен в силу нескольких причин. За исключением плана системного тестирования, все остальные документы существуют в нескольких экземплярах. Шаблон гарантирует непротиворечивость формы и содержания этих независимых, но в то же время родственных по содержанию документов. Чем лучше документ вписывается в тот или иной шаблон, тем меньше усилий затрачивает разработчик при создании его многочисленных экземпляров. Использование шаблонов к тому же упрощает процесс контроля, поскольку каждый документ исполнен в одном и том же стиле, что позволяет быстро ознакомиться с его конкретным содержанием.

В схеме плана тестирования, рекомендованного стандартом IEEE и представленного на рис. 3.14, приводится список основных разделов, которые должны присутствовать в каждом плане тестирования, независимо от его уровня. Мы хотим рассмотреть те из них, которые имеют наибольшее значение в среде разработки интерактивного, итеративного объектно-ориентированного программного обеспечения. В последующих планах тестирования мы не будем называть разделы в строгом соответствии с этой схемой, однако включим в них информацию об основных требованиях. Приводимые ниже пункты плана тестирования имеют особую важность:

- *Свойства, не подлежащие тестированию* — для тестирования на уровне классов. В этом разделе содержится информацию HIT-анализа (Hierarchical Incremental Analysis — Иерархический инкрементный анализ, см. главу 7). Эта информация содержит данные о свойствах, которые уже подверглись тестированию и не нуждаются в повторном тестировании, и о свойствах, разработка которых запланирована на последующие итерации или на последующие комплектующие модули.
- *Критерии приостановки тестирования и требования для его возобновления* — тестирование приостанавливается, когда показатели тестирования выходят на недопустимый уровень, т.е. когда количество обнаруженных за один час ошибок опускается ниже критерия, установленного для данного раздела, что, собственно, и приводит к приостановке тестирования. Этот раздел имеет особую важность для проектов, разрабатываемых в итеративном режиме. Обычно мы определяем один набор критериев для циклов разработки на ранней стадии и другой набор — для более поздних циклов разработки. В случае итеративной разработки проекта за критерий возобновления принимается возврат цикла разработки в контрольную точку.
- *Риски и сопряженность признаков* — риск в этом контексте обозначает потенциальные проблемы, сопряженные с проведением тестирования. В их число входят возможные ошибки в правильных ответах, представленных в виде крупных наборов данных, а также вероятность того, что на различных платформах будут получены разные результаты, хотя только лишь некоторые случаи подверглись тестированию.

1.0 Введение
2.0 Тестовые элементы
3.0 Тестируемые свойства
4.0 Свойства, не подлежащие тестированию (за один цикл)
5.0 Стратегия тестирования и подход
    5.1 Синтаксис
    5.2 Описание выполняемых функций
    5.3 Аргументация в пользу испытаний
    5.4 Ожидаемые выходные данные
    5.5 Специально оговариваемые исключения
    5.6 Зависимости
    5.7 Критерии успешного/неудачного выполнения тестирования
6.0 Критерии успешного прохождения/неудачи полного тестового цикла
7.0 Критерии вхождения/критерии выхода
8.0 Критерии приостановки испытаний и требования к их возобновлению
9.0 Тестовые комплектующие узлы/средства передачи данных о состоянии
10.0 Задачи тестирования
11.0 Требования к аппаратным средствам и программному обеспечению
12.0 Обнаружение проблем и назначение ответственных за коррекцию
13.0 Кадровое обеспечение и необходимость обучения/назначения на должности
14.0 График работ по тестированию
15.0 Риски и сопряженность признаков
16.0 Официальное одобрение

Рисунок 3.14. *Шаблон стандартного типа тестирования IEEE829.*

План тестирования проекта

Назначением этого проекта является формулирование стратегии тестирования, которая должна применяться к проекту. Он должен определить этапы разработки, на которых следует выполнять тестирование, частоту выполнения процедур тестирования, а также указать, на кого возложена ответственность за этот вид деятельности.

План тестирования проекта может быть независимым документом, он может также быть включен в общий план работ по проекту или в план обеспечения качества. Поскольку формат этого документа часто меняется, а его содержание достаточно гибкое, мы приводим здесь примеры таблиц, в которых такого рода информация представляется в обобщенном виде.

В таблице на рис. 3.15 дается общая информация о видах деятельности, которые необходимо выполнить, показано, как часто должен выполняться тот или иной вид деятельности, и указан ответственный за данную фазу тестирования.

Более подробная информация по всем показателям представлена в детализированном плане для данного уровня.

Вторая таблица, которая приводится на рис. 3.16, назначает каждой фазе специальную стратегию тестирования.

Шаблон плана тестирования проекта – часть I

Проект _____

Ответственный исполнитель _____

Уровень тестирования	Виды деятельности	Частота тестирования	Ответственный исполнитель
Компонент	Выбор тестовых примеров	По мере готовности компонентов	Разработчик компонента
	Написание классов PACT		Тестировщик компонента
Интеграция	Выбор тестовых примеров	До выпуска комплектующего модуля	Группа интеграции модулей
Система	Выбор тестовых примеров по случаям использования	До сдачи какого-либо варианта системы внешнему заказчику	Отдел обеспечения качества
	Построение классов PAST		

Рисунок 3.15. *Шаблон плана тестирования проекта — часть I*

Шаблон плана тестирования проекта – часть II

Уровень тестирования	Стратегия тестирования	Уровень профиля пользователя	Критерий покрытия
Компонент		Высокий	
		Средний	
		Низкий	
Интеграция		Высокий	
		Средний	
		Низкий	
Система		Высокий	
		Средний	
		Низкий	

Рисунок 3.16. *Шаблон плана тестирования проекта — часть II*

Мы предложим несколько стратегий тестирования в соответствующих главах с тем, чтобы вы смогли сделать свой выбор. В этой таблице зафиксированы также проектные стандарты адекватного тестирования для каждого уровня риска в рамках указанных выше трех фаз.

План тестирования компонентов

Назначение плана тестирования компонентов заключается в определении общей стратегии тестирования и конкретных тестовых случаев, которые будут использоваться для тестирования того или иного компонента. Для каждого компонента составляется один план, в котором должна быть обоснована необходимость автономного тестирования этого компонента. Мы приводим здесь шаблон, который успешно применяли в своей работе. В этот шаблон включены два типа руководящей информации: проектные критерии и проектные процедуры. Они включены для того, чтобы служить ненавязчивым напоминанием, а также во избежание необходимости разработки плана тестирования компонентов, в котором обобщается вся информация о тестировании компонентов данного проекта. Проектными критериями являются согласованные стандарты, определяющие, насколько тщательно должно быть выполнено тестирование каждого компонента. Например, проектные критерии могут требовать 100%-ного тестирования постусловий методов модификатора. Эти критерии должны содержать дальнейшие подробности о критериях покрытия, определенных в плане тестирования проекта. Проектные процедуры выявляют методы, которые были выбраны как лучшие способы решения той или иной конкретной задачи. Например, построение класса PACT (см. главу 7) для каждого компонента, который будет протестирован в проектной процедуре. Эти процедуры содержат подробности о стратегиях тестирования, которые были зафиксированы в плане тестирования проектов.

Ниже приводятся краткие комментарии по каждому разделу шаблона. Сам шаблон находится на рис. 3.17. Мы не будем описывать разделы, в которых просто фиксируется такая информация, как, например, наименование того или иного компонента. Курсивом выделяется фактически внесенная в шаблон информация.

Цели построения класса. Разработчик должен заменить этот параграф списком целей построения компонента, упорядоченным по возрастанию приоритетов. Например, компонент представляет собой элемент базовой структуры приложения и предназначен для использования в качестве высокоуровневой абстракции, от которой порождаются более конкретные варианты.

Руководящие требования по проверке. Проектные критерии: *Все 100% программных продуктов, связанных с наиболее ответственными компонентами, подвергаются тестированию. 75% компонентов, связанных с менее ответственными компонентами, подвергаются тестированию. Библиотечные компоненты проходят дополнительную проверку качества.* Проектная процедура: *Анализ рисков используется для упорядочивания компонентов класса по приоритетам с целью их проверки и тестирования.*

Построение и сохранение тестовых наборов. Разработчик должен заменить этот параграф информацией о:

- результатах применения HIT-анализа и подробностях использования процесса PACT для построения классов тестового драйвера (см. главу 7)
- запланированном предельном сроке разработки тестовых случаев
- спецификации тестового драйвера
- относительном числе тестовых случаев в каждой категории и приоритетах каждой из этих трех категорий.

Функциональные тестовые случаи. Разработчик должен заменить этот параграф информацией о:

- тестовых случаях, разработанных в соответствии со спецификацией
- методе инвариантов классов
- сколько различных «типов» объектов подлежат тестированию. Количество этих типов зависит от начального состояния объекта.

Структурные тестовые случаи. Разработчик должен заменить этот параграф информацией о:

- тестовых случаях, разработанных для покрытия программных кодов, и о процессе ревизии программных кодов
- как использовать требуемые инструментальные средства, поддерживающие покрытие тестами объектов тестирования.

Тестовые случаи, построенные на базе состояний классов. Разработчик должен заменить этот параграф информацией о представлении состояний класса. Воспользуйтесь диаграммами состояний, если таковые имеются.

Тестовые случаи для тестирования взаимодействия компонентов. Разработчик должен заменить этот параграф информацией о том, какие сообщения подвергаются тестированию на базе процесса OATS (Orthogonal Array Testing System — система тестирования с использованием ортогональной матрицы), обеспечивающего выборку образцов (см. главу 6).

План тестирования случаев использования

Назначение этого плана заключается в описании, какие испытания на уровне системы строятся на базе одного конкретного случая использования. При помощи ссылок эти планы включаются в план комплексных испытаний и в план системного тестирования. На рис. 3.18, 3.19 и 3.20 представлены фрагменты шаблона плана тестирования случаев использования. Другие фрагменты приводятся в главе 9.

Планы тестирования можно строить в соответствии с модульным принципом по одному и тому же образцу как зависимости между «частичными» случаями использования.

Шаблон плана тестирования компонента

Название компонента _____

Ответственный исполнитель _____

Имя разработчика _____

Задания на тестирование класса

Проектная процедура: Тестовые случаи предназначены для тестирования компонентов в соответствии с возложенными на данный компонент задачами.

Руководящие требования по проверке

Проектные критерии: Все 100% программных продуктов, связанных с наиболее ответственными компонентами, подвергаются тестированию. 75% компонентов, связанных с менее ответственными компонентами, подвергаются тестированию. Библиотечные компоненты проходят дополнительную проверку качества.

Особо важный компонент приложения _____

Менее важный компонент приложения _____

Библиотечный компонент _____

Проектные критерии: Анализ рисков используется для упорядочивания по приоритетам компонентов класса с целью их проверки и тестирования.

Построение и сохранение тестовых наборов

Проектная процедура: Алгоритм HIT используется для определения, прогон каких тестовых случаев следует осуществить для данного компонента.

Проектная процедура: В обязанности разработчика входит подготовка тестовых наборов в структуре, затребованной проектом.

Проектные критерии: Для каждого компонента имеется класс тестового драйвера, который содержит тестовые случаи для тестируемого компонента.

Проектные критерии: Для каждого компонента имеются тестовые методы, которые представляют функциональные, структурные тестовые случаи, а также тестовые случаи, предназначенные для тестирования взаимодействия компонентов.

Функциональные тестовые случаи

Проектные критерии: Выполнить тестовые случаи для проверки каждого постусловия каждого метода. Выполнить также проверку инвариантов классов как часть тестового случая.

Проектные критерии: Выполнить прогон тестовых случаев при наиболее важных значениях каждого из параметров.

Структурные тестовые случаи

Проектные критерии: Выполнить тестовые случаи, покрывающие каждую строку программного кода каждого метода.

Критерий анализа рисков по отношению к тестируемому компоненту _____

Тестовые случаи на базе состояний классов

Проектные критерии: Выполнить прогон тестовых случаев, которые покрывают каждый переход из одного представления в другое.

Тестовые случаи для тестирования взаимодействия компонентов

Проектная процедура: Выполнить прогон тестовых случаев, проверяющих каждый контракт между компонентами. Воспользуйтесь процессом OATS для выбора тестовых случаев с целью их последующего прогона.

Рисунок 3.17. *Шаблон плана тестирования компонента*

Шаблон плана тестирования случая использования - часть 1

Наименование случая использования _____

Ответственный исполнитель: Ответственность за отладку классов несет разработчик, которому *принадлежит* компонент.

Имя разработчика: _____

Часть 1. Внешний вид и компоновка (повторяется для каждого экранного изображения/отчета)

1.1	Все необходимые поля данных присутствуют.	Да	Нет
1.2	Элементы представлены в корректном порядке.	Да	Нет
1.3	Неописанные поля отсутствуют.	Да	Нет
1.4	Начальные системные значения по умолчанию выбраны правильно.	Да	Нет
1.5	(Только для экранов) Обход по клавише табуляции осуществляется в корректном порядке.	Да	Нет
1.6	(Только для экранов) Клавиши ускоренного доступа работают правильно.	Да	Нет
1.7	(Только для экранов) Возможен доступ к полям в произвольном порядке.	Да	Нет
1.8	(Только для экранов) Меню упорядочены правильно.	Да	Нет
1.9	(Только для экранов) Соответствующие функции меню активны.	Да	Нет
1.10	(Только для экранов) Объекты реагируют корректно (на события типа двойного щелчка и т.п.).	Да	Нет

Стандарты для тестовых случаев

1. Для каждого «обязательного» поля должны быть предусмотрены тестовые случаи, в которых значение для такого поля отсутствует.
2. Для каждого граничного значения должны быть предусмотрены тестовые случаи, в которых используется это конкретное значение. Должны быть также тестовые случаи, которые используют значения из каждого класса эквивалентности из интервала между этими граничными значениями.
3. Для каждого поля с перечислимым типом данных, кроме флажков, должны существовать тестовые случаи для выявления полей с неправильными значениями или пустых полей.
4. Для каждого поля типа даты должны предусматриваться тестовые случаи, выявляющие поля с неправильными значениями или пустые поля. Однако, пока явно заданные ограничения на случаи использования отсутствуют, проверка граничных значений невозможна. Даты могут быть заданы неправильно в силу особенностей формата либо из-за нарушений бизнес-правил.
5. Для каждого сценария, в котором имеет место выборка данных, должен быть предусмотрен тестовый случай, в котором все поля пусты, за исключением единственного, в котором данные присутствуют. Такой тестовый случай должен быть выбран для каждого поля, которое может редактироваться пользователем. Для тех полей, которые принимают типизированные значения, например, поле имени, следует предусмотреть два тестовых случая. Например, должны проверяться как полное имя, так и частичное имя.

Часть 3 — Безопасность/Целостность

Тестовая программа указывает уровень защиты пользователя и выполняет тестирование каждого уровня	Да	Нет
Каждое поле точно обнаруживает неправильно заданные типы данных и их обработку	Да	Нет
Система выдает сообщение об исправлении ошибки и предупреждающее сообщение о возникновении необычной ситуации.	Да	Нет
Выходные данные успешно продублированы на всех серверах.	Да	Нет

1. Тесты, которые должны быть успешно пройдены, прежде чем станет возможным выполнение таких сценариев:
2. Тесты, которые необходимо выполнить, если неудачно завершится выполнение следующих сценариев:
3. Тесты, которые должны быть выполнены после успешного завершения следующих тестов:

Рисунок 3.18. *Шаблон плана тестирования случая использования — часть 1*

Шаблон плана тестирования случая использования - часть 2

Наименование случая использования _____

Ответственный исполнитель: Ответственность за отладку классов несет разработчик, которому *принадлежит* компонент.

Имя разработчика: _____

Часть 4 — Сценарии тестирования

В этом разделе содержатся инструкции тестировщикам о том, как проводить каждое тестирование. Эти инструкции берутся из раздела «Система отвечает...» каждого сценария случая использования и из таблиц, построенных в разделе 2 шаблона случая использования, который на рис. 3.18 не показан по причине экономии пространства книги, зато его можно найти на рис. 9.7.

Сценарий #

Обеспечить выполнение предусловий для тестов

Тип пользователя: любой

Пользователь обладает специальными полномочиями, необходимыми для выполнения операции	Да	Нет
Система вводит в действие соответствующее управление доступом	Да	Нет
Появляется соответствующее сообщение об ошибке при попытке несанкционированного доступа	Да	Нет
Система отвергает систематические попытки несанкционированного доступа	Да	Нет

Потребности конфигурации

Необходимые тестовые базы данных и таблицы на месте	Да	Нет

Выполнить следующий сценарий:

Выполнить оценку результатов

Общие результаты тестирования:

Получен ожидаемый результат.	Да	Нет
Значение каждого связанного поля вычислено правильно.	Да	Нет
Выпадение из синхронизации набора связанных полей вызывает появление сообщения об ошибке.	Да	Нет
Появилось требуемое сообщение об ошибке.	Да	Нет
Сообщение дает точное и однозначное описание проблемы.	Да	Нет
Система восстановилась соответствующим образом после выдачи сообщения об ошибке.	Да	Нет
Сохранение было невозможным до исправления ошибок.	Да	Нет
Программа соответствует стандартам по производительности.	Да	Нет

Рисунок 3.19. *Шаблон плана тестирования случая использования — часть 2*

Шаблон плана тестирования случая использования - часть 3

Наименование случая использования _____

Ответственный исполнитель: Ответственность за отладку классов несет разработчик, которому *принадлежит* компонент.

Имя разработчика: _____

Часть 5 — Краткое описание тестирования случая использования

Информация общего характера о тестировании как о виде деятельности (ответственный персонал, место проведения испытаний и прочее):

_____

_____

_____

Описание аппаратных средств, использованных для прогона тестов (модель принтера, тип сетевого соединения и прочее): _____

_____

Неописанное спецификацией поведение, которое имело место: _____

_____

Отклонение от описанных спецификацией тестовых процедур:

_____

_____

_____

Требуется повторное тестирование. Да Нет

Утверждаю: _____

Рисунок 3.20. *Шаблон плана тестирования случая использования — часть 3*

Модели случаев использования могут быть структурированы таким же образом, как и диаграммы классов. Отношения **includes** и **extends** обеспечивают средства декомпозиции случаев использования в «частичные» случаи использования, как показано в главе 2. Частичные случаи использования объединяются с помощью указанных выше отношений, образуя при этом «комплексные» случаи использования.

Мы выделяем три уровня случаев использования: высокоуровневые, сквозные системные и функциональные подслучаи использования. Высокоуровневые случаи использования — это абстрактные случаи использования, которые могут служить основой для их расширения до сквозных системных случаев использования. Функциональные подслучаи использования агрегируются в сквозные системные случаи использования. Мы написали фактические сценарии тестирования на языке сценариев, входящем в состав инструментальных средств тестирования, которые используют отношение генерации/специализации между высокоуровневыми и сквозными системными случаями использования. Эти тестовые сценарии также собирают в единое целое фрагменты сценариев тестирова-

ния из функциональных подслучаев использования. Обладая тремя рассмотренными уровнями, проекты становятся более управляемыми, а тестовые сценарии — более модульными.

В проекте, для которого это служило шаблоном, различаются два «вида» случаев использования: функциональные и статистические случаи использования. Функциональные случаи использования изменяют данные, поддерживаемые системой тем или иным способом. Статистические случаи использования получают информацию от системы, после чего обобщают и форматируют ее для последующего представления пользователю. Эти различия привели к различному количеству сеансов тестирования безопасности и устойчивости. В собственных проектах могут обнаружиться и другие группы случаев использования.

План комплексных испытаний

План комплексных испытаний особенно важен в итеративной среде разработки. Некоторые специальные наборы функциональных средств появляются раньше других. С добавлением таких комплектующих модулей система постепенно приобретает свои очертания. Одно из осложнений, вытекающее из такого стиля развития системы, состоит в том, что план комплексных испытаний меняет свой характер на протяжении жизни проекта гораздо чаще, чем план тестирования компонентов или план системного тестирования. Компоненты, которые интегрируются в систему на ранних стадиях как комплектующие модули, могут непосредственно не поддерживать функциональные средства конечного пользователя, и в силу этого обстоятельства ни один из случаев использования не может служить источником тестовых случаев. На этой стадии наилучшим таким источником являются планы тестирования компонентов для агрегированных компонентов. Такие планы используются для завершения плана тестирования компонента, в задачу которого входит интегрирование этих объектов. После того как соответствующее число комплектующих модулей будет включено в систему, функциональные возможности интегрированного программного обеспечения начнут приходить в соответствие с поведением системного уровня. На этом временном этапе наилучшим источником тестовых случаев являются планы тестирования случаев использования.

В обоих ситуациях тестовые случаи выбираются в зависимости от того, насколько такой тестовый случай нуждается в том, чтобы поведение оказывало воздействие на все интегрируемые модули. Небольшое, локально действующее поведение должно быть уже отлажено. Это означает, что тестирование должно быть более сложным и более всесторонним, нежели обычное тестирование компонента. В интегрированной по всем правилам объектно-ориентированной системе сушествуют определенные объекты, которые перекрывают некоторое число других объектов, входяших в ее состав. Выбор тестов в соответствии с планами тестирования этих компонентов довольно часто позволяет получить тестовые случаи для комплексных испытаний.

В связи с такой зависимостью планов комплексных испытаний от тестовых случаев, мы не приводим для него какого-то специального шаблона. Его формат совпадает с форматом шаблона для плана системных испытаний в том смысле, что он отражает определенную комбинацию индивидуальных планов тестирования, образующую план комплексных испытаний при внедрении конкретного комплектующего модуля.

План системных испытаний

План системных испытаний представляет собой документ, который обобщает планы тестирования отдельных случаев использования и предоставляет информацию о дополнительных видах тестирования, которые могут проводиться на системном уровне. В каждой из глав, в которых рассматриваются методы тестирования, будет описано тестирование жизненного цикла как один из методов, который может применяться как на системном уровне, так и на уровне отдельных компонентов.

Для последующего использования здесь вводится таблица (рис. 3.21), которая отображает планы тестирования случаев использования на системные испытания. Большая часть информации, которая требуется форматом IEEE планов тестирования, будет предоставлена планами тестирования индивидуальных случаев использования.

План системных испытаний

Номер случая использования	Номер тестового случая	Обоснование выбора

Рисунок 3.21. *План системных испытаний*

Итерации в планировании

Итерации в процессе разработки влияют на то, как выполняется планирование. Изменения в требованиях, предъявляемых к программному продукту или к комплектующему модулю, требуют, по меньшей мере, того, чтобы планы периодически подвергались ревизии. Во многих случаях они требуют изменений. Чтобы облегчить выполнение таких модификаций в цикле, мы поддерживаем матрицу обнаруживаемости неисправностей.

Если организация-разработчик получает требования в традиционной форме, мы строим матрицу отображения требований на случаи использования. Часто это просто крупноформатная электронная таблица, в которой по вертикальной оси откладываются идентификаторы требований, а по горизонтальной оси —

идентификаторы случаев использования. Вхождение в какую-либо ячейку таблицы указывает на то, что функциональные возможности случая использования определяются или ограничиваются соответствующим требованием.

Кроме того, поддерживается вторая матрица, при помощи которой каждый случай использования соотносится с набором пакетов классов. Вхождение в ту или иную ячейку соответствующей таблицы означает, что в пакете имеются классы, которые используются при реализации соответствующего случая использования. Когда случай использования меняется, владельцы пакета получают сообщение об этом. Они проверяют функциональные средства, которые они предоставляют, и вносят требуемые изменения в программный код. Это вызывает последовательность изменений на нескольких уровнях тестовых случаев и, возможно, в планах тестирования.

Планирование

Объем работ по планированию зависит от нескольких факторов:

- от того, как часто повторно используется шаблоны
- от того, сколько приходится затрачивать усилий, чтобы превратить шаблон в реальный завершенный план
- от того, какие требуются трудозатраты, чтобы внести изменения в существующий план.

Каждое из этих значений требует назначения базовой линии с тем, чтобы, отталкиваясь от нее, можно было получить реальные оценки.

Система показателей тестирования

Система показателей тестирования предусматривает измерения, которые дают информацию, необходимую для оценки отдельных методов тестирования и всего процесса тестирования. Показатели тестирования используются также как исходные данные для планирования, например, оценок трудозатрат для проведения тестирования. Чтобы получить окончательные значения показателей такого рода, потребуются меры, обеспечивающие возможность измерения покрытия и сложности программного продукта и позволяющие построить базу для систем показателей эффективности и производительности.

Покрытие — это термин, применяющийся в тестировании, который показывает, какие компоненты были затронуты тестовыми случаями. В этой книге мы рассмотрим целый ряд различных мер покрытий во время обсуждения методов тестирования. При этом основное внимание будет уделено следующим моментам:

- **Покрытие программных кодов** — какие строки программных кодов выполнялись.
- **Покрытие постусловий** — какие постусловия учитывались при выполнении тестовых случаев.
- **Покрытия элементов моделей** — какие классы и отношения модели были использованы в тестовых случаях.

Система показателей покрытия выражается в терминах программного *продукта*, подвергающегося тестированию, но не в терминах *процесса*, который используется для его отладки. Это дает нам основание, в соответствии с которым можно говорить, с «какой тщательностью» программный продукт был испытан. Например, рассмотрим ситуацию, когда один разработчик использует каждый логический оператор из каждого постусловия в качестве источника тестовых случаев, в то время как другой использует из постусловий только те, которые рассчитаны на благоприятные стечение обстоятельств (или т.н. условия «солнечного дня»[8]). Второй разработчик не выполняет тестирования так тщательно, как первый, поскольку тестированием покрывается только некоторая часть операторов рассматриваемого постусловия.

Покрытие в сочетании со сложностью может стать надежной базой для вычисления трудозатрат, необходимых для тестирования программного продукта. Другими словами, по мере усложнения программного обеспечения, становится все труднее достичь заданного уровня покрытия. Сложность можно измерять несколькими способами:

- количеством и сложностью методов конкретного класса
- количеством строк программного кода
- количеством динамических связей.

Собирая данные о рабочих характеристиках программных продуктов на протяжении достаточно продолжительного промежутка времени, проекты или компании могут установить некоторую базовую линию, отталкиваясь от которой можно строить планы относительно каждого нового проекта.

Процесс тестирования результативен, если он позволяет выявить ошибки. Он эффективен, если позволяет выявлять ошибки с минимально возможными затратами ресурсов. Мы рассмотрим пару способов, позволяющих получить данные об обеих этих характеристиках. **Показатель число отказов/затраты человеко-часов разработчиков** определяет выход процесса тестирования, в то время как **показатель затраты человеко-часов разработчиков/число неисправностей** представляет собой меру стоимости процесса тестирования. Эти значения зависят от того, какие инструментальные средства используются для построения тестов, равно как и для достижения требуемого уровня покрытия, откуда следует, что каждой компании потребуется определить базовую линию для собственного процесса тестирования и сбора фактических данных о его эффективности, прежде чем использовать эти значения для целей планирования.

Оценка результативности процесса тестирования производится путем сбора данных на протяжении всего цикла разработки проекта. Рассмотрим случай ошибки, привнесенной в приложение во время разработки. Чем скорее этот дефект будет обнаружен, тем более результативным является процесс тестирова-

[8] Конструкция «солнечного дня» — это ожидаемый результат, не принимающий во внимание ошибочные состояния, которые могут вызвать исключительную ситуацию или возврат кода ошибки.

ния. Эффективность этого процесса тестирования измеряется длительностью интервала времени от момента, когда эта неисправность была внесена в программный продукт, и до стадии, на которой она была обнаружена во всех ее проявлениях. По-настоящему эффективный процесс тестирования обнаруживает каждый дефект на той же стадии разработки, на какой она была внесена. Если же дефекты, привнесенные во время разработки, остаются необнаруженными до стадии тестирования программных кодов, то методы тестирования, применяемые во время разработки системы; должны быть модифицированы таким образом, чтобы они могли отыскивать и те типы неисправностей, которые ранее вовремя не обнаруживались.

Резюме

В мире, пронизанном духом конкуренции, в котором сложное программное обеспечение продается по цене каких-то 99.95 долл. США, а компании выплачивают поощрительные премии любому служащему, который привлечет к работе нового специалиста по программного обеспечению, планирование становится исключительно важным видом деятельности. Основная трудность заключается в том, чтобы найти правильное сочетание между затратами времени на разработку планов и их документирование и затратами времени на производство программного продукта.

Мы описали последовательность шаблонов, которые позволяют сократить затраты времени на процесс планирования. Эти документы образуют иерархическую структуру, благодаря которой становится возможным дальнейшее сокращение объемов документации, необходимой для выполнения адекватной работы. Процесс планирования позволяет добиться успеха, если разработчики, являющиеся движущейся силой проекта, находят планирование полезным. Планирование становится в лучшем случае формальным, если разработчики будут выполнять его лишь «для проформы».

Упражнения

3-1 Назовите документы и модели, которые необходимо подготовить для проектируемой системы. Продумайте способы выдачи необходимой информации об отсутствующих компонентах.

3-2 Расставьте по приоритетам требования к своему проекту и к результирующему программному продукту.

3-3 Продолжите разработку плана тестирования, построив таблицу, в которой содержится список всех «продуктов», доступных для тестирования. Вторая колонке таблицы должна быть отведена под список методов тестирования, которые планируется задействовать при тестировании программного продукта. На текущий момент эта колонка остается пустой. Ваш проект итеративный? Он инкрементальный? Если он итеративный, следует предусмотреть колонку, в которой обобщаются данные применения различных методов во время каждой итерации. Если процесс инкрементальный, должны существовать отдельные таблицы для каждого комплектующего модуля системы, поскольку они разрабатываются независимо друг от друга.

3-4 Какой уровень риска вы присвоите классам **Sprite**, **PuckSuply**, **Puck** и **Velocity** игры «Кирпичики»?

Глава 4

Тестирование аналитических и проектных моделей

- Вы хотите научиться проверять семвнтику моделей, нвписанных нв языке UML? См. раздел «Основные понятия целенаправленной проверки»
- Ввм нужно инициироввть сеанс проверки? См. рвздел «Оргвнизвция целенапрввленной инспекционной деятельности»
- Вам нужен метод тестирования модели на рвсширяемость? См. рвздел «Модели для тестирования дополнительных свойств»

Разработчики обычно моделируют создаваемые программные продукты, поскольку модель помогает им глубже понять решаемую задачу, а также в силу того, что они помогают успешно справляться со сложностью разрабатываемых ими систем. Модели аналитической и проектной информации в конечном итоге будут использованы для того, чтобы направить усилия по реализации проекта в правильное русло. Если модели обладают высоким качеством, они вносят неоценимый вклад в реализацию проекта, но если в них содержатся ошибки, они настоль же вредны. В этой главе рассматривается понятие **целенаправленной проверки**, представляющей собой метод расширенной проверки моделей по мере их создания и контроля этих моделей на предмет соответствия требованиям проекта. Основной недостаток стандартных ревизий заключаются в том, что центр тяжести проверки они переносят главным образом на то, что уже *имеется* (в модели), а не на то, что в ней *должно быть*. В ревизиях не предусмотрены средства систематического обнаружения того, чего не хватает в программном продукте. Даже проверки Фагана (Fagan) [Faga86], в которых применяются контрольные списки для обеспечения дальнейшей детализации процесса, не предусматривают средств для определения, чего в модели не хватает.

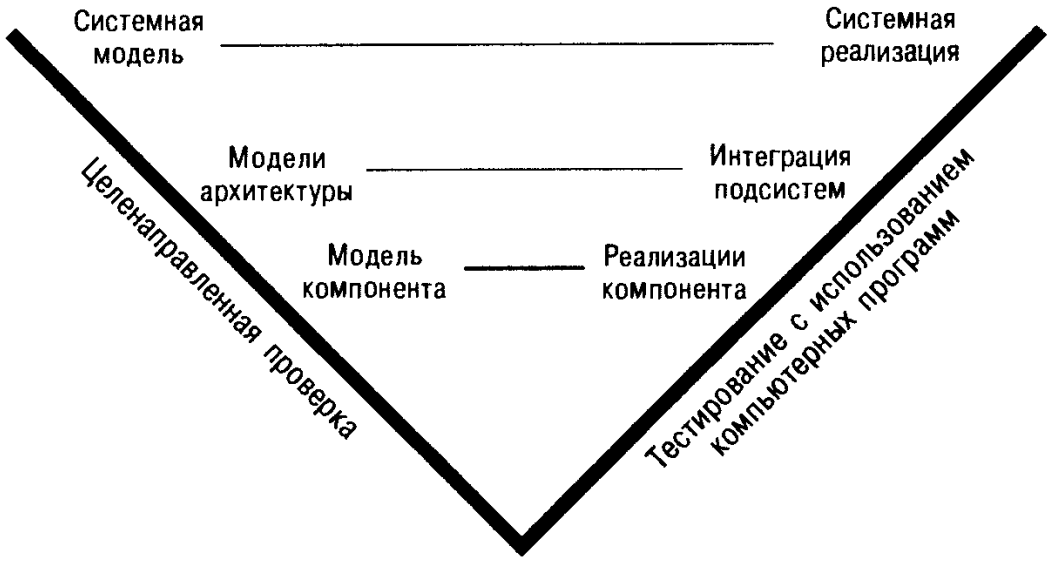

Рисунок 4.1. *Новая V-образная модель*

Целенаправленная проверка применяет перспективу тестирования на очень ранних стадиях процесса разработки. Традиционно тестирование начиналось на уровне реализации модулей и продолжалось по мере того, как сегменты программных кодов объединялись во фрагменты большего размера до тех пор, пока не вся система не оказывалась готовой к тестированию. В этой главе мы начнем «системное тестирование» на стадии, когда «система» представлена в виде аналитической или проектной информации. Новая версия традиционной V-образной модели тестирования, показанная на рис. 4.1, соотносит повторяющиеся применения целенаправленных проверок с различными уровнями тестирования.

Целенаправленная проверка требует затрат дорогостоящих ресурсов и внимания со стороны персонала, работающего над проектом. Есть ли практический смысл в ее проведении? Исследования показывают, что отношение затрат на обнаружение и устранение неисправностей на ранней стадии процесса разработки к затратам по их обнаружению и устранению на этапе компиляции или системных испытаний колеблется в широких пределах. Например, устранение ошибки, обнаруженной во время системных испытаний, может стоить на два порядка дороже, нежели устранение той же ошибки на этапе анализа. Таким образом, даже умеренные затраты усилий на тестирование модели могут принести большую экономию.

Обзор

Рассмотрим простой пример использования метода целенаправленной проверки. Прежде всего, потребуется определиться с декорациями – мы находимся на исходных стадиях разработки игры «Кирпичики». Коллектив разработчиков построил диаграмму классов для этапа проектирования, показанную на рис. 2.18, и ряд других диаграмм, таких как, например, диаграмма состояний (см. рис. 2.19) и диаграмма последовательности сообщений (см. рис. 2.20). Мы уже готовы начать написание программных кодов, однако хотим убедиться в корректности проектной модели и не тратить уйму времени на кодирование неправильных определений.

Начнем с того, что назначим группу проверки программ. В эту группу входят оба автора этой книги, работавшие над моделью, системный тестировщик нашей компании и координатор, в роли которого выступает специалист, знакомый с технологическим процессом компании. Тестировщик будет разрабатывать набор тестовых случаев по диаграмме случаев использования. Будучи разработчиками, мы покажем, как классы в модели проекта манипулируют каждым случаем использования. Координатор будет определять границы проверки, составлять расписание сеансов целенаправленных проверок, распределять материалы, обеспечивать продвижение сеансов и составлять сводный отчет.

Подготавливая сеанс, координатор определяет границы проверки, выбирая область проверки и глубину тестируемой информации. Область проверки определяется набором случаев использования. В рассматриваемом случае область проверки покрывает все случаи использования, следовательно, и все приложение. Глубина охвата определяется выбором уровней включения в иерархиях композиции. В случае игры «Кирпичики» мы не проверяем объектов, которые агрегированы в объекте **BricklesView**. Вместо этого внимание будет сосредоточено на тех из них, которые представляют состояние сеанса игры в любой заданный момент времени в объекте **BricklesDoc.**

Тестировщик пишет тестовые случаи на базе случаев использования, представленных на рис. 2.11. Остановим свой выбор на одном из тестовых случаев, показанном на рис. 4.2. Перед тем как встретиться на рабочем совещании, проектировщики завершают построение контрольной таблицы проектной модели, показанной на рис. 4.3. Эта задача решается каждым разработчиком индивидуально. При этом требуется, чтобы каждый разработчик сравнивал диаграмму классов из модели анализа, представленной на рис. 2.13, с диаграммой классов проектной модели. В завершение координатор посылает уведомление о предстоящем совещании вместе с копиями модели на бумаге или с URL-адресом ее Web-версии.

В протоколе испытаний, поступающем от группы целенаправленной проверки, отражены все проблемы, обнаруженные во время символического выполнения тестовых случаев. Что касается рассматриваемого здесь тестового случая, то предполагается, что на данной стадии проект еще не прошел испытаний. Протокол испытаний отразит факт невозможности отыскания способа завершения символического выполнения тестового случая в данной точке алгоритма. Мы не хотим смешивать тестирование с отладкой, тем не менее, поскольку известно, в какой точке символическое выполнение было остановлено, это должно найти свое отражение в протоколе. В протокол испытаний включена диаграмма последовательности сообщений, использованная для регистрации выполненных тестов (см. рис. 4.4).

Случай использоаания: Игрок заввршает сеанс игры «Кирпичики», выбирая в меню File элемент Quit.
Прадусловия: Игрок начал сеанс игры «Кирпичики», первмещал лопатку и даже умудрился разбить несколько кирпичвй.
Входные данные теста: Игрок выбрал элемент Quit.
Ожидаемые выходные данные: Все игровые действия замораживаются, а окно игры исчвзает с экрана.

Рисунок 4.2. *Тестовый случай №1*