ВЫСШЕЕ ОБРАЗОВАНИЕ

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

А. В. Игнатьев

ЛАНЬ

E.LANBOOK.COM

А. В. ИГНАТЬЕВ

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

УЧЕБНОЕ ПОСОБИЕ

Издание третье, стереотипное

ЛАНЬ

· САНКТ-ПЕТЕРБУРГ · МОСКВА · КРАСНОДАР ·
· 2023 ·

УДК 004.45
ББК 32.972я73

И 26 **Игнатьев А. В.** Тестирование программного обеспечения : учебное пособие для вузов / А. В. Игнатьев. — 3-е изд., стер. — Санкт-Петербург : Лань, 2023. — 56 с. : ил. — Текст : непосредственный.

ISBN 978-5-507-45425-9

Учебное пособие посвящено вопросам анализа, планирования, проведения тестовых испытаний и оценки качества программного обеспечения на всех стадиях его жизненного цикла. Является методическим обеспечением выполнения лабораторных работ для студентов вузов, обучающихся по направлениям «Информационные системы и технологии» и «Программная инженерия».

УДК 004.45
ББК 32.972я73

Обложка
П. И. ПОЛЯКОВА

ВВЕДЕНИЕ

Настоящее учебно-методическое пособие предназначено для проведения лабораторных работ по дисциплине «Тестирование и отладка программного обеспечения» для студентов, обучающихся по направлениям подготовки 09.03.02 «Информационные системы и технологии», 09.03.04 «Программная инженерия».

Учебно-методическое пособие посвящено вопросам анализа, планирования, проведения тестовых испытаний и оценки качества программного обеспечения на всех стадиях его жизненного цикла.

Подробно раскрывается тема классификации видов тестирования. Изучаются основы планирования тестирования, разработки рабочей тестовой документации, поиска и описания дефектов, оценки качества и документирования результатов тестирования.

При разработке учебно-методического пособия использованы источники, список которых приведен в конце работы.

ЛАБОРАТОРНАЯ РАБОТА № 1. ТЕСТИРОВАНИЕ ТРЕБОВАНИЙ

Цель: изучить критерии качества требований, выполнить тестирование требований к программному обеспечению.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчёт и ответить на контрольные вопросы.

Теоретические сведения

Качество программного обеспечения во многом зависит от качества сформированных требований, так как требования к программному продукту являются базой для разработки и последующего тестирования.

Тестирование требований является необходимой и очень важной процедурой, которая в дальнейшем поможет оптимизировать работу команды и избежать недопонимания, а также позволяет понять, можно ли в принципе выполнить данные требования — с точки зрения времени, ресурсов и бюджета.

Форма представления, степень детализации и перечень полезных свойств требований зависят от уровней и типов требований.

Тестируются требования, описывающие функциональность проекта, пользовательский, аппаратный, программный интерфейсы, критерии эффективности, риски, критерии безопасности и корректности системы.

Тестирование требований выполняется на предмет их соответствия критериям качества требований, перечисленным далее.

Завершённость (completeness). Требование является полным и законченным с точки зрения представления в нём всей необходимой информации, ничто не пропущено по соображениям «это и так всем понятно».

Типичные проблемы с завершённостью:

— отсутствуют нефункциональные составляющие требования или ссылки на соответствующие нефункциональные требования (например: «пароли должны храниться в зашифрованном виде», а каков алгоритм шифрования?);

— указана лишь часть некоторого перечисления (например: «экспорт осуществляется в форматы PDF, PNG и т. д.», а что следует понимать под «и т. д.»?);

— приведённые ссылки неоднозначны (например: «см. выше» вместо «см. раздел 123.45.b»).

Атомарность, единичность (atomicity). Требование является атомарным, если его нельзя разбить на отдельные требования без потери завершённости и оно описывает одну и только одну ситуацию.

Типичные проблемы с атомарностью:

– в одном требовании фактически содержатся несколько независимых (например: «кнопка “Restart” не должна отображаться при остановленном сервисе, окно “Log” должно вмещать не менее 20 записей о последних действиях пользователя»: здесь в одном предложении описаны совершенно разные элементы интерфейса в совершенно разных контекстах);

— требование допускает разночтение в силу грамматических особенностей языка (например: «если пользователь подтверждает заказ и редактирует заказ или откладывает заказ, должен выдаваться запрос на оплату»: здесь описаны три разных случая, и это требование стоит разбить на три отдельных требования во избежание путаницы). Такое нарушение атомарности часто влечёт за собой возникновение противоречивости;

— в одном требовании объединено описание нескольких независимых ситуаций (например: «когда пользователь входит в систему, должно отображаться приветствие; когда пользователь вошёл в систему, должно отображаться имя пользователя; когда пользователь выходит из системы, должно отображаться прощание»: все эти три ситуации заслуживают того, чтобы быть описанными отдельными и более детальными требованиями).

Непротиворечивость, последовательность (consistency). Требование не должно содержать внутренних противоречий и противоречий другим требованиям и документам.

Типичные проблемы с непротиворечивостью:

— противоречия внутри одного требования (например: «после успешного входа в систему пользователя, не имеющего права входить в систему...»: а как пользователь вошёл в систему, если не имел такого права?);

— противоречия между двумя и более требованиями, между таблицей и текстом, рисунком и текстом, требованием и прототипом и т. д. (например: «712.a Кнопка “Close” всегда должна быть красной» и «36452.x Кнопка “Close” всегда должна быть синей»: так всё же красной или синей?);

— использование неверной терминологии или использование разных терминов для обозначения одного и того же объекта или явления (например: «в случае если разрешение окна составляет менее 800×600...»: разрешение есть у экрана, у окна есть размер).

Недвусмысленность (unambiguousness, clearness). Требование описано без использования жаргона, неочевидных аббревиатур и расплывчатых формулировок и допускает только однозначное объективное понимание. Требование атомарно в плане невозможности различной трактовки сочетания отдельных фраз.

Типичные проблемы с недвусмысленностью:

— использование терминов или фраз, допускающих субъективное толкование (например: «приложение должно поддерживать передачу больших объёмов данных»: насколько «больших»?). Вот лишь небольшой перечень слов и выражений, которые можно считать верными признаками двусмысленности: адекватно, быть способным, легко, обеспечивать, как минимум, быть способным, эффективно, своевременно, применимо, если возможно, будет определено позже, по мере необходимости, если это целесообразно, но не ограничиваясь, быть способно, иметь возможность, нормально, минимизировать, максимизировать, оптимизировать, быстро, удобно, просто, часто, обычно, большой, гибкий, устойчивый, по последнему слову техники, улучшенный, результативно;

— использование неочевидных или двусмысленных аббревиатур без расшифровки (например: «доступ к ФС осуществляется посредством системы прозрачного шифрования» и «ФС предоставляет возможность фиксировать сообщения в их текущем состоянии с хранением истории всех изменений»: ФС здесь обозначает файловую систему или какой-нибудь «Фиксатор Сообщений»?);

— формулировка требований из соображений, что нечто должно быть всем очевидно (например: «Система конвертирует входной файл из формата PDF в выходной файл формата PNG», и при этом автор считает совершенно очевидным, что имена файлов система получает из командной строки, а многостраничный PDF конвертируется в несколько PNG-файлов, к именам которых добавляется «page-1», «page-2» и т. д.). Эта проблема перекликается с нарушением корректности.

Выполнимость (feasibility). Требование технологически выполнимо и может быть реализовано в рамках бюджета и сроков разработки проекта.

Типичные проблемы с выполнимостью:

— «озолочение» (gold plating) — требования, которые крайне долго и/или дорого реализуются и при этом практически бесполезны для конечных пользователей (например: «настройка параметров для подключения к базе данных должна поддерживать распознавание символов из жестов, полученных с устройств трёхмерного ввода»);

— технически нереализуемые на современном уровне развития технологий требования (например: «анализ договоров должен выполняться с применением искусственного интеллекта, который будет выносить однозначное корректное заключение о степени выгоды от заключения договора»);

— в принципе нереализуемые требования (например: «система поиска должна заранее предусматривать все возможные варианты поисковых запросов и кэшировать их результаты»).

Обязательность, нужность (obligation) и актуальность (up-to-date). Если требование не является обязательным к реализации, оно должно быть просто исключено из набора требований. Если требование нужное, но «не очень важное», для указания этого факта используется указание приоритета. Также должны быть исключены (или переработаны) требования, утратившие актуальность.

Типичные проблемы с обязательностью и актуальностью:

— требование было добавлено «на всякий случай», хотя реальной потребности в нём не было и нет;

— требованию выставлены неверные значения приоритета по критериям важности и/или срочности;

— требование устарело, но не было переработано или удалено.

Прослеживаемость (traceability). Прослеживаемость бывает вертикальной и горизонтальной. Вертикальная позволяет соотносить между собой требования на различных уровнях требований, горизонтальная позволяет соотносить требование с тест-планом, тест-кейсами, архитектурными решениями и т. д.

Для обеспечения прослеживаемости часто используются специальные инструменты по управлению требованиями и/или матрицы прослеживаемости.

Типичные проблемы с прослеживаемостью:

— требования не пронумерованы, не структурированы, не имеют оглавления, не имеют работающих перекрёстных ссылок;

— при разработке требований не были использованы инструменты и техники управления требованиями;

— набор требований неполный, носит обрывочный характер с явными «пробелами».

Модифицируемость (modifiability). Это свойство характеризует простоту внесения изменений в отдельные требования и в набор требований. Можно говорить о наличии модифицируемости в том случае, если при доработке требований искомую информацию легко найти, а её изменение не приводит к нарушению иных описанных в этом перечне свойств.

Типичные проблемы с модифицируемостью:

— требования неатомарны (см. «атомарность») и непрослеживаемы, а потому их изменение с высокой вероятностью порождает противоречивость;

— требования изначально противоречивы, в такой ситуации внесение изменений (не связанных с устранением противоречивости) только усугубляет ситуацию, увеличивая противоречивость и снижая прослеживаемость;

— требования представлены в неудобной для обработки форме (например, не использованы инструменты управления требованиями, и в итоге команде приходится работать с десятками огромных текстовых документов).

Проранжированность по важности, стабильности, срочности (ranked for importance, stability, priority). Важность характеризует зависимость успеха проекта от успеха реализации требования. Стабильность характеризует вероятность того, что в обозримом будущем в требование не будет внесено никаких изменений. Срочность определяет распределение во времени усилий проектной команды по реализации того или иного требования.

Типичные проблемы с проранжированностью состоят в её отсутствии или неверной реализации и приводят к следующим последствиям.

Проблемы с проранжированностью по важности повышают риск неверного распределения усилий проектной команды, направления усилий на второстепенные задачи и конечного провала проекта из-за неспособности продукта выполнять ключевые задачи с соблюдением ключевых условий.

Проблемы с проранжированностью по стабильности повышают риск выполнения бессмысленной работы по совершенствованию, реализации и тестированию требований, которые в самое ближайшее время могут претерпеть кардинальные изменения (вплоть до полной утраты актуальности).

Проблемы с проранжированностью по срочности повышают риск нарушения желаемой заказчиком последовательности реализации функциональности и ввода этой функциональности в эксплуатацию.

Корректность (correctness) и проверяемость (verifiability). Фактически эти свойства вытекают из соблюдения всех вышеперечисленных (или можно сказать, что они не выполняются, если нарушено хотя бы одно из вышеперечисленных). В дополнение можно отметить, что проверяемость подразумевает возможность создания объективного тест-кейса (тест-кейсов), однозначно показывающего,

что требование реализовано верно и поведение приложения в точности соответствует требованию.

К типичным проблемам с корректностью также можно отнести:

— опечатки (особенно опасны опечатки в аббревиатурах, превращающие одну осмысленную аббревиатуру в другую, также осмысленную, но не имеющую отношения к некоему контексту, такие опечатки крайне сложно заметить);

— наличие неаргументированных требований к дизайну и архитектуре;

— плохое оформление текста и сопутствующей графической информации;

— грамматические, пунктуационные и иные ошибки в тексте;

— неверный уровень детализации (например, слишком глубокая детализация требования на уровне бизнес-требований или недостаточная детализация на уровне требований к продукту);

— требования к пользователю, а не к приложению (например: «пользователь должен быть в состоянии отправить сообщение»: мы не можем влиять на состояние пользователя).

Техники тестирования требований

1. Одной из наиболее активно используемых техник анализа требований является просмотр, или рецензирование. Данная техника может быть реализована в форме:

— беглого просмотра (показ автором своей работы коллеге; самый быстрый, самый дешёвый и наиболее широко используемый вид просмотра);

— технического просмотра (выполняется группой специалистов, каждый из которых представляет свою область знаний: просматриваемый продукт не может считаться достаточно качественным, пока хотя бы у одного просматривающего остаются замечания);

— формальной инспекции (структурированный, систематизированный и документируемый подход к анализу документации, для выполнения которого привлекается большое количество специалистов, само выполнение занимает достаточно много времени, и потому этот вариант просмотра используется достаточно редко).

2. Следующей техникой тестирования и повышения качества требований является использование такой техники выявления требований, как формулировка вопросов. Если хоть что-то в требованиях вызывает непонимание или подозрение — задавайте вопросы.

3. Хорошее требование является проверяемым, а значит, должны существовать объективные способы определения того, верно ли реа-

лизовано требование. Продумывание чек-листов или даже полноценных тест-кейсов в процессе анализа требований позволяет определить, насколько требование проверяемо. Помимо использования для тестирования требований в дальнейшем такие чек-листы и тест-кейсы могут составить основу тестовой документации.

4. Чтобы увидеть общую картину требований целиком, очень удобно использовать рисунки, схемы, диаграммы, интеллект-карты и т. д. Графическое представление удобно одновременно своей наглядностью и краткостью (например, UML-диаграмма классов, занимающая один экран, может быть описана несколькими десятками страниц текста).

5. Исследование поведения и прототипирование. Прототипирование часто является следствием создания графического представления и анализа поведения системы. С использованием специальных инструментов можно очень быстро сделать наброски пользовательских интерфейсов, оценить применимость тех или иных решений и даже создать заготовку для дальнейшей разработки, если окажется, что реализованное в прототипе устраивает заказчика.

Практическое задание

Команде разработки, в которой вы работаете, от постоянного клиента (ОАО «Рога и Копыта») поступил заказ на создание онлайн-ресурса, позволяющего пользователям покупать фильмы, арендовать фильмы на короткий период или смотреть любые видео сервиса в неограниченном количестве по подписке.

Со стороны клиента вам были предоставлены первичные требования к программному продукту, представленные ниже. Требования к программному продукту будут полностью формализованы в виде документа спецификации требований после их согласования с командой разработки (в вашем лице).

Вам необходимо ознакомиться с представленным списком требований и составить список вопросов и уточнений, которые позволят вам максимально улучшить качество требований (в соответствии со свойствами качества требований).

Коммуникация с клиентом затруднена, и каждое согласование требований занимает длительное время. Вопросы с непонятной клиенту формулировкой повлекут за собой увеличение срока согласования требований. Помимо этого, если вы ожидаете, что некоторые ответы клиента повлекут за собой еще больше вопросов, — постарайтесь задать эти вопросы сразу же.

Требования к программному обеспечению

Общие требования:

1. Поддерживается работа во всех основных браузерах — Chrome, Yandex, IE, Edge, Opera, Safari.

2. Цветовая схема веб-ресурса полностью соответствует цветам компании «Рога и Копыта».

3. Пользователи в любой точке мира не должны испытывать проблем с соединением и получать отклик от системы в пределах 100 мс с момента действия.

4. Удовлетворяются требования к хранению и обработке пользовательских данных, регламентированные Россией, Евросоюзом и США.

5. Должна гарантироваться работа системы при использовании веб-сервиса 1 миллионом пользователей.

6. Поддерживается русский и английский языки, а также имеется возможность добавления дополнительных локализаций при необходимости.

Навигация:

7. На главной странице, в меню навигации, находятся кнопки перехода между разделами (главная страница, выбор жанров фильмов, выбор жанров сериалов, история просмотров, библиотека пользователя) и кнопка регистрации/авторизации/настроек пользователя.

8. В верхней части расположена область популярных фильмов, где отображается слайд-шоу карточек популярных фильмов с поправкой на предпочтения пользователя.

9. Ниже области популярных фильмов располагается таблица с карточками фильмов, разделённых по жанрам.

10. По нажатию на КФ открывается страница деталей выбранного фильма.

11. На странице деталей фильма отображается следующая информация: название фильма, краткое описание, обложка фильма, трейлер к фильму, список серий (для сериалов), предложения похожих фильмов, кнопки покупки фильма и аренды фильма на 30 дней.

12. На странице покупки отображается форма ввода платежной информации и поддерживается функция сохранения платежной информации для её дальнейшего быстрого ввода.

13. Если у пользователя куплен или арендован фильм или если у пользователя приобретена платная подписка, то на странице деталей фильма также отображается видеоплеер, в котором пользователь может просмотреть фильм.

Видеоплеер:

14. В видеоплеере есть кнопки воспроизведения/паузы, переключатель качества видео, громкости видео и отображения субтитров.

15. Поддерживается функция буферизации видео.

16. По завершении просмотра видео пользователю отображается возможность оценить фильм, а также предлагаются для просмотра несколько похожих фильмов.

17. При возникновении ошибки соединения видео ставится на паузу, отображается спиннер и сообщение об ошибке.

18. При исправлении ошибки видео сразу же продолжает воспроизведение.

Содержание отчёта:

1. Цель работы.
2. Отчёт по тестированию спецификации.
3. Выводы по работе.

Контрольные вопросы

1. Какие выделяют критерии качества требований?
2. Какие требования являются завершёнными?
3. Перечислите основные проблемы, связанные с завершённостью требований.
4. Какие требования являются атомарными?
5. Перечислите основные проблемы, связанные с атомарностью требований.
6. Какие требования являются непротиворечивыми?
7. Перечислите основные проблемы, связанные с непротиворечивостью требований.
8. Какие требования являются недвусмысленными?
9. Перечислите основные проблемы, связанные с недвусмысленностью требований.
10. Какие требования являются выполнимыми?
11. Перечислите основные проблемы, связанные с выполнимостью требований.
12. Какие требования являются обязательными, нужными и актуальными?
13. Перечислите основные проблемы, связанные с обязательностью и актуальностью требований.
14. Какие требования являются прослеживаемыми?
15. Перечислите основные проблемы, связанные с прослеживаемостью требований.
16. Какие требования являются модифицируемыми?

17. Перечислите основные проблемы, связанные с модифицируемостью требований.

18. Какие требования считаются проранжированными по важности?

19. Какие требования считаются проранжированными по стабильности?

20. Какие требования считаются проранжированными по срочности?

21. Перечислите основные проблемы, связанные с проранжированностью требований по важности.

22. Перечислите основные проблемы, связанные с проранжированностью требований по стабильности.

23. Перечислите основные проблемы, связанные с проранжированностью требований по срочности.

24. Какие требования являются корректными?

25. Перечислите основные проблемы, связанные с корректностью требований.

26. Какие существуют техники тестирования требований? В чём особенности каждой их них?

ЛАБОРАТОРНАЯ РАБОТА № 2. РАЗРАБОТКА МОДУЛЬНЫХ И ИНТЕГРАЦИОННЫХ ТЕСТОВ

Цель: изучить принципы разработки модульных и интеграционных тестов.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчёт и ответить на контрольные вопросы.

Теоретические сведения

Модульное тестирование

Модульное тестирование — это тестирование программы на уровне отдельно взятых модулей, функций или классов. Цель модульного тестирования состоит в выявлении локализованных в модуле ошибок в реализации алгоритмов, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования. Модульное тестирование проводится по принципу «белого ящика», то есть основывается на знании внутренней структуры программы, и часто включает те или иные методы анализа покрытия кода.

Модульное тестирование обычно подразумевает создание вокруг каждого модуля определённой среды, включающей заглушки для всех интерфейсов тестируемого модуля. Некоторые из них могут использоваться для подачи входных значений, другие — для анализа результатов, присутствие третьих может быть продиктовано требованиями, накладываемыми компилятором и сборщиком.

На уровне модульного тестирования проще всего обнаружить дефекты, связанные с алгоритмическими ошибками и ошибками кодирования алгоритмов, типа работы с условиями и счётчиками циклов, а также с использованием локальных переменных и ресурсов. Ошибки, связанные с неверной трактовкой данных, некорректной реализацией интерфейсов, совместимостью, производительностью и т. п. обычно пропускаются на уровне модульного тестирования и выявляются на более поздних стадиях тестирования.

Именно эффективность обнаружения тех или иных типов дефектов должна определять стратегию модульного тестирования, то есть расстановку акцентов при определении набора входных значений. У организации, занимающейся разработкой программного обеспечения, как правило, имеется историческая база данных (Repository) раз-

работок, хранящая конкретные сведения о разработке предыдущих проектов: о версиях и сборках кода (build), зафиксированных в процессе разработки продукта, о принятых решениях, допущенных просчётах, ошибках, успехах и т. п. Проведя анализ характеристик прежних проектов, подобных заказанному организации, можно предохранить новую разработку от старых ошибок, например определив типы дефектов, поиск которых наиболее эффективен на различных этапах тестирования.

В данном случае анализируется этап модульного тестирования. Если анализ не дал нужной информации, например в случае проектов, в которых соответствующие данные не собирались, то основным правилом становится поиск локальных дефектов, у которых код, ресурсы и информация, вовлечённые в дефект, характерны именно для данного модуля. В этом случае на модульном уровне ошибки, связанные, например, с неверным порядком или форматом параметров модуля, могут быть пропущены, поскольку они вовлекают информацию, затрагивающую другие модули (а именно: спецификацию интерфейса), в то время как ошибки в алгоритме обработки параметров довольно легко обнаруживаются.

Являясь по способу исполнения структурным тестированием или тестированием «белого ящика», модульное тестирование характеризуется степенью, в которой тесты выполняют или покрывают логику программы (исходный текст).

Интеграционное тестирование

Интеграционное тестирование — это тестирование части системы, состоящей из двух и более модулей. Основная задача интеграционного тестирования — поиск дефектов, связанных с ошибками в реализации и интерпретации интерфейсного взаимодействия между модулями.

С технологической точки зрения интеграционное тестирование является количественным развитием модульного, поскольку так же, как и модульное тестирование, оперирует интерфейсами модулей и подсистем и требует создания тестового окружения, включая заглушки (Stub) на месте отсутствующих модулей. Основная разница между модульным и интеграционным тестированием состоит в целях, то есть в типах обнаруживаемых дефектов, которые в свою очередь определяют стратегию выбора входных данных и методов анализа. В частности, на уровне интеграционного тестирования часто применяются методы, связанные с покрытием интерфейсов, например вызовов функций или методов, или анализ использования интер-

фейсных объектов, таких как глобальные ресурсы, средства коммуникаций, предоставляемых операционной системой.

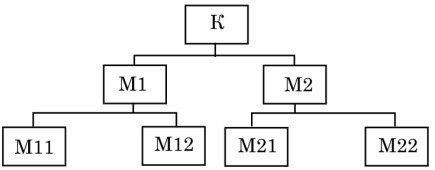

Рис. 2.1

Пример структуры комплекса программ

На рисунке 2.1 приведена структура комплекса программ К, состоящего из оттестированных на этапе модульного тестирования модулей М1, М2, М11, М12, М21, М22. Задача, решаемая методом интеграционного тестирования, — тестирование межмодульных связей, реализующихся при исполнении программного обеспечения комплекса К. Интеграционное тестирование использует модель «белого ящика» на модульном уровне. Поскольку тестировщику текст программы известен с детальностью до вызова всех модулей, входящих в тестируемый комплекс, применение структурных критериев на данном этапе возможно и оправдано.

Интеграционное тестирование применяется на этапе сборки модульно оттестированных модулей в единый комплекс. Известны два метода сборки модулей:

1) монолитный, характеризующийся одновременным объединением всех модулей в тестируемый комплекс;

2) инкрементальный, характеризующийся пошаговым (помодульным) наращиванием комплекса программ с пошаговым тестированием собираемого комплекса. В инкрементальном методе выделяют две стратегии добавления модулей:

- «сверху вниз» и соответствующее ему нисходящее тестирование;
- «снизу вверх» и, соответственно, восходящее тестирование.

Практическое задание

1. Создать приложение «Калькулятор», поддерживающее следующие функции:

- базовые функции (сложение, вычитание, умножение, деление);
- возведение в степень;
- извлечение квадратного корня;
- факториал выбранного числа;
- операции с памятью (сохранение числа в память, чтение из памяти, добавление к числу в памяти, очистка памяти).

2. Каждую операцию написать в виде отдельной функции.

3. Покрыть каждую из операций модульными тестами.

4. Убедиться, что тесты успешно проходятся в созданном приложении.

Содержание отчёта:

1. Цель работы.

2. Код приложения, включая код всех тестов.

3. Выводы по работе.

Контрольные вопросы

1. Дайте определение модульного тестирования.

2. Какова цель модульного тестирования?

3. По какому принципу проводится модульное тестирование?

4. Что включает в себя среда, создаваемая вокруг каждого модуля?

5. Какие дефекты проще всего обнаружить на уровне модульного тестирования?

6. Какие ошибки обычно пропускаются на уровне модульного тестирования и выявляются на более поздних стадиях тестирования?

7. Что определяет стратегию модульного тестирования?

8. Дайте определение интеграционного тестирования.

9. Почему с технологической точки зрения интеграционное тестирование является количественным развитием модульного?

10. В чём заключается основная разница между модульным и интеграционным тестированием?

11. На каком этапе применяется интеграционное тестирование?

12. Охарактеризуйте известные методы сборки модулей.

ЛАБОРАТОРНАЯ РАБОТА № 3. СОСТАВЛЕНИЕ ТЕСТ-ПЛАНА

Цель: изучить принципы разработки тест-плана.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчёт и ответить на контрольные вопросы.

Теоретические сведения

Тест-план

Тест-план (test plan) — документ, описывающий цели, подходы, ресурсы и график запланированных тестовых активностей. Он определяет объекты тестирования, свойства для тестирования, задания, ответственных за задания, степень независимости каждого тестировщика, тестовое окружение, метод проектирования тестов, определяет используемые критерии входа и критерии выхода и причины их выбора, а также любые риски, требующие планирования на случай чрезвычайных обстоятельств.

К низкоуровневым задачам планирования в тестировании относятся:

— оценка объёма и сложности работ;

— определение необходимых ресурсов и источников их получения;

— определение расписания, сроков и ключевых точек;

— оценка рисков и подготовка превентивных контрмер;

— распределение обязанностей и ответственности;

— согласование работ по тестированию с деятельностью участников проектной команды, занимающихся другими задачами.

Качественный тест-план обладает большинством свойств качественных требований, а также расширяет их набор следующими пунктами:

— реалистичность (запланированный подход реально выполним);

— гибкость (качественный тест-план не только является модифицируемым с точки зрения работы с документом, но и построен таким образом, чтобы при возникновении непредвидённых обстоятельств допускать быстрое изменение любой из своих частей без нарушения взаимосвязи с другими частями);

— согласованность с общим проектным планом и иными отдельными планами (например, планом разработки).

Тест-план создаётся в начале проекта и дорабатывается по мере необходимости на протяжении всего времени жизни проекта при уча-

стии наиболее квалифицированных представителей проектной команды, задействованных в обеспечении качества. Ответственным за создание тест-плана, как правило, является ведущий тестировщик («тест-лид»).

В общем случае тест-план включает в себя следующие разделы:

— цель (purpose) — предельно краткое описание цели разработки приложения (частично это напоминает бизнес-требования, но здесь информация подаётся в ещё более сжатом виде и в контексте того, на что следует обращать первостепенное внимание при организации тестирования и повышения качества);

— области, подвергаемые тестированию (features to be tested) — перечень функций и/или нефункциональных особенностей приложения, которые будут подвергнуты тестированию. В некоторых случаях здесь также приводится приоритет соответствующей области;

— области, не подвергаемые тестированию (features not to be tested) — перечень функций и/или нефункциональных особенностей приложения, которые не будут подвергнуты тестированию. Причины исключения той или иной области из списка тестируемых могут быть самыми различными — от предельно низкой их важности для заказчика до нехватки времени или иных ресурсов. Этот перечень составляется, чтобы у проектной команды и иных заинтересованных лиц было чёткое единое понимание, что тестирование таких-то особенностей приложения не запланировано — такой подход позволяет исключить появление ложных ожиданий и неприятных сюрпризов;

— тестовая стратегия (test strategy) и подходы (test approach) — описание процесса тестирования с точки зрения применяемых методов, подходов, видов тестирования, технологий, инструментальных средств и т. д.;

— критерии (criteria) — этот раздел включает следующие подразделы:

- приёмочные критерии, критерии качества (acceptance criteria) — любые объективные показатели качества, которым разрабатываемый продукт должен соответствовать с точки зрения заказчика или пользователя, чтобы считаться готовым к эксплуатации;
- критерии начала тестирования (entry criteria) — перечень условий, при выполнении которых команда приступает к тестированию. Наличие этого критерия страхует команду от бессмысленной траты усилий в условиях, когда тестирование не принесёт ожидаемой пользы;

• критерии приостановки тестирования (suspension criteria) — перечень условий, при выполнении которых тестирование приостанавливается. Наличие этого критерия также страхует команду от бессмысленной траты усилий в условиях, когда тестирование не принесёт ожидаемой пользы;

• критерии возобновления тестирования (resumption criteria) — перечень условий, при выполнении которых тестирование возобновляется (как правило, после приостановки);

• критерии завершения тестирования (exit criteria) — перечень условий, при выполнении которых тестирование завершается. Наличие этого критерия страхует команду как от преждевременного прекращения тестирования, так и от продолжения тестирования в условиях, когда оно уже перестаёт приносить ощутимый эффект;

— ресурсы (resources). В данном разделе тест-плана перечисляются все необходимые для успешной реализации стратегии тестирования ресурсы, которые в общем случае можно разделить на:

• программные ресурсы;

• аппаратные ресурсы;

• человеческие ресурсы;

• временные ресурсы;

• финансовые ресурсы (во многих компаниях финансовые ресурсы могут быть представлены отдельным документом, так как являются конфиденциальной информацией).

— расписание (test schedule) — фактически это календарь, в котором указано, что и к какому моменту должно быть сделано. Особое внимание уделяется так называемым ключевым точкам (milestones), к моменту наступления которых должен быть получен некий значимый ощутимый результат;

— роли и ответственность (roles and responsibility) — перечень необходимых ролей (например, «ведущий тестировщик», «эксперт по оптимизации производительности») и область ответственности специалистов, выполняющих эти роли;

— оценка рисков (risk evaluation) — перечень рисков, которые с высокой вероятностью могут возникнуть в процессе работы над проектом. По каждому риску даётся оценка представляемой им угрозы и приводятся варианты выхода из ситуации;

— документация (documentation) — перечень используемой тестовой документации с указанием, кто и когда должен её готовить и кому передавать;

— метрики (metrics) — числовые характеристики показателей качества, которые могут включать в себя описание способов оценки и анализа результата. На этот раздел, как правило, формируется множество ссылок из других разделов тест-плана.

Метрики в тестировании являются настолько важными, что их необходимо рассмотреть отдельно.

Метрики могут быть как прямыми (не требуют вычислений), так и расчётными (вычисляются по формуле). Типичные примеры прямых метрик — количество разработанных тест-кейсов, количество найденных дефектов и т. д. В расчётных метриках могут использоваться как совершенно тривиальные, так и довольно сложные формулы.

В тестировании существует большое количество общепринятых метрик, многие из которых могут быть собраны автоматически с использованием инструментальных средств управления проектами. Например:

— процентное отношение (не) выполненных тест-кейсов ко всем имеющимся;

— процентный показатель успешного прохождения тест-кейсов;

— процентный показатель заблокированных тест-кейсов;

— плотность распределения дефектов;

— эффективность устранения дефектов;

— распределение дефектов по важности и срочности;

и т. д.

При формировании отчётности важно учитывать не только текущее значение метрики, но и её динамику во времени, которую очень удобно изображать графически (что тоже могут выполнять автоматически многие средства управления проектами).

Таким образом, метрики являются мощнейшим средством сбора и анализа информации. И вместе с тем здесь кроется опасность: ни при каких условиях нельзя допускать ситуации «метрик ради метрик», когда инструментальное средство собирает уйму данных, вычисляет множество чисел и строит десятки графиков, но… никто не понимает, как их трактовать.

И наконец стоит упомянуть про так называемые метрики покрытия.

Покрытие (coverage) — процентное выражение степени, в которой исследуемый элемент (coverage item) затронут соответствующим набором тест-кейсов.

Самыми простыми представителями метрик покрытия можно считать:

— метрику покрытия требований (требование считается покрытым, если на него ссылается хотя бы один тест-кейс);

— метрику плотности покрытия требований (учитывается, сколько тест-кейсов ссылается на несколько требований);

— метрику покрытия классов эквивалентности (анализируется, сколько классов эквивалентности затронуто тест-кейсами);

— метрику покрытия граничных условий (анализируется, сколько значений из группы граничных условий затронуто тест-кейсами);

— метрики покрытия кода модульными тест-кейсами. Таких метрик очень много, но вся их суть сводится к выявлению некоей характеристики кода (количество строк, ветвей, путей, условий и т. д.) и определению, какой процент представителей этой характеристики покрыт тест-кейсами.

Метрик покрытия настолько много, что даже в ISTQB-глоссарии дано определение полутора десяткам таковых.

Практическое задание

Вы является руководителем команды QA в компании, занимающейся разработкой мобильного мессенджера. Вашей задачей является написание тест-плана, максимально полно описывающего процесс тестирования программного продукта.

В качестве программного продукта, для которого вы пишете тест-план, можете взять любой коммерческий мессенджер (Whatsapp, ICQ или что-то еще).

Если для описания каких-то пунктов плана вам не будет хватать информации (например, вам требуется придумать сотрудников, которые занимаются тестированием или сослаться на несуществующий документ требований) — описывайте гипотетический (но реалистичный) сценарий, как если бы эта информация была вам доступна.

Содержание отчёта:

1. Цель работы.
2. Разработанный тест-план.
3. Выводы по работе.

Контрольные вопросы

1. Дайте определение тест-плана.
2. Назовите низкоуровневые задачи планирования в тестировании.
3. Назовите свойства качественного тест-плана.
4. На каком этапе проекта создаётся тест-план?
5. Кто является ответственным за создание тест-плана?

6. Какие разделы в общем случае включает в себя тест-план?

7. Какие следующие подразделы включает в себя раздел «Критерии»?

8. На какие виды, в общем случае, можно разделить ресурсы, необходимые для успешной реализации стратегии тестирования?

9. Дайте определение метрики.

10. Что такое прямые и расчётные метрики? Приведите примеры.

11. Дайте определение покрытия.

12. Перечислите самые простые метрики покрытия.

ЛАБОРАТОРНАЯ РАБОТА № 4. ПОДГОТОВКА ЧЕК-ЛИСТОВ, ТЕСТ-КЕЙСОВ, НАБОРОВ ТЕСТ-КЕЙСОВ

Цель: изучить принципы разработки чек-листов, тест-кейсов, наборов тест-кейсов.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчёт и ответить на контрольные вопросы.

Теоретические сведения

Чек-лист

Чек-лист чаще всего представляет собой обычный и привычный нам список, который может быть:

— списком, в котором последовательность пунктов не имеет значения (например, список значений некоего поля);

— списком, в котором последовательность пунктов важна (например, шаги в краткой инструкции);

— структурированным (многоуровневым) списком (вне зависимости от учёта последовательности пунктов), что позволяет отразить иерархию идей.

Важно понять, что нет и не может быть никаких запретов и ограничений при разработке чек-листов — главное, чтобы они помогали в работе. Иногда чек-листы могут даже выражаться графически (например, с использованием ментальных карт или концепт-карт), хотя традиционно их составляют в виде многоуровневых списков.

Поскольку в разных проектах встречаются однотипные задачи, хорошо продуманные и аккуратно оформленные чек-листы могут использоваться повторно, чем достигается экономия сил и времени.

Для того чтобы чек-лист был действительно полезным инструментом, он должен обладать рядом следующих важных свойств.

Логичность. Чек-лист пишется не «просто так», а на основе целей и для того, чтобы помочь в достижении этих целей. К сожалению, одной из самых частых и опасных ошибок при составлении чек-листа является превращение его в свалку мыслей, которые никак не связаны друг с другом.

Последовательность и структурированность. Со структурированностью всё достаточно просто — она достигается за счёт оформления чек-листа в виде многоуровневого списка. Что до последовательности, то даже в том случае, когда пункты чек-листа не опи-

сывают цепочку действий, человеку всё равно удобнее воспринимать информацию в виде неких небольших групп идей, переход между которыми является понятным и очевидным (например, сначала можно прописать идеи простых позитивных тест-кейсов, потом идеи простых негативных тест-кейсов, потом постепенно повышать сложность тест-кейсов, но не стоит писать эти идеи вперемешку).

Полнота и неизбыточность. Чек-лист должен представлять собой аккуратную «сухую выжимку» идей, в которых нет дублирования (часто появляется из-за разных формулировок одной и той же идеи), и в то же время ничто важное не упущено.

Правильно создавать и оформлять чек-листы также помогает восприятие их не только как хранилища наборов идей, но и как «требования для составления тест-кейсов». Эта мысль приводит к пересмотру и переосмыслению свойств качественных требований в применении к чек-листам.

Тест-кейс

Тест-кейс (test case) — набор входных данных, условий выполнения и ожидаемых результатов, разработанный с целью проверки того или иного свойства или поведения программного средства.

Под тест-кейсом также может пониматься соответствующий документ, представляющий формальную запись тест-кейса.

Если у тест-кейса не указаны входные данные, условия выполнения и ожидаемые результаты и/или не ясна цель тест-кейса — это плохой тест-кейс (иногда он не имеет смысла, иногда его и вовсе невозможно выполнить).

Иногда термин «test case» на русский язык переводят как «тестовый случай». Это вполне адекватный перевод, но из-за того, что «тест-кейс» короче произносить, наибольшее распространение получил именно этот вариант.

Набор тест-кейсов называется тестовым набором (test suite).

Высокоуровневый тест-кейс (high level test case) — тест-кейс без конкретных входных данных и ожидаемых результатов.

Как правило, ограничивается общими идеями и операциями, схож по своей сути с подробно описанным пунктом чек-листа. Достаточно часто встречается в интеграционном тестировании и системном тестировании, а также на уровне дымового тестирования. Может служить отправной точкой для проведения исследовательского тестирования или для создания низкоуровневых тест-кейсов.

Низкоуровневый тест-кейс (low level test case) — тест-кейс с конкретными входными данными и ожидаемыми результатами.

Представляет собой «полностью готовый к выполнению» тест-кейс и вообще является наиболее классическим видом тест-кейсов. Начинающих тестировщиков чаще всего учат писать именно такие тесты, так как прописать все данные подробно — намного проще, чем понять, какой информацией можно пренебречь, при этом не снизив ценность тест-кейса.

Спецификация тест-кейса (test case specification) — документ, описывающий набор тест-кейсов (включая их цели, входные данные, условия и шаги выполнения, ожидаемые результаты) для тестируемого элемента (test item, test object).

Тест-сценарий (test scenario, test procedure specification, test script) — документ, описывающий последовательность действий по выполнению теста (также известен как «тест-скрипт»).

Цель написания тест-кейсов

Наличие же тест-кейсов позволяет:

— структурировать и систематизировать подход к тестированию (без чего крупный проект почти гарантированно обречён на провал);

— вычислять метрики тестового покрытия (test coverage metrics) и принимать меры по его увеличению (тест-кейсы здесь являются главным источником информации, без которого существование подобных метрик теряет смысл);

— отслеживать соответствие текущей ситуации плану (сколько примерно понадобится тест-кейсов, сколько уже есть, сколько выполнено из запланированного на данном этапе количества и т. д.);

— уточнить взаимопонимание между заказчиком, разработчиками и тестировщиками (тест-кейсы зачастую намного более наглядно показывают поведение приложения, чем это отражено в требованиях);

— хранить информацию для длительного использования и обмена опытом между сотрудниками и командами (или как минимум — не пытаться удержать в голове сотни страниц текста);

— проводить регрессионное тестирование и повторное тестирование (которые без тест-кейсов было бы вообще невозможно выполнить);

— повышать качество требований (мы это уже рассматривали: написание чек-листов и тест-кейсов — хорошая техника тестирования требований);

— быстро вводить в курс дела нового сотрудника, недавно подключившегося к проекту.

Жизненный цикл тест-кейса

Для тест-кейса речь скорее идёт о наборе состояний (рис. 4.1), в которых он может находиться (жирным шрифтом отмечены наиболее важные состояния).

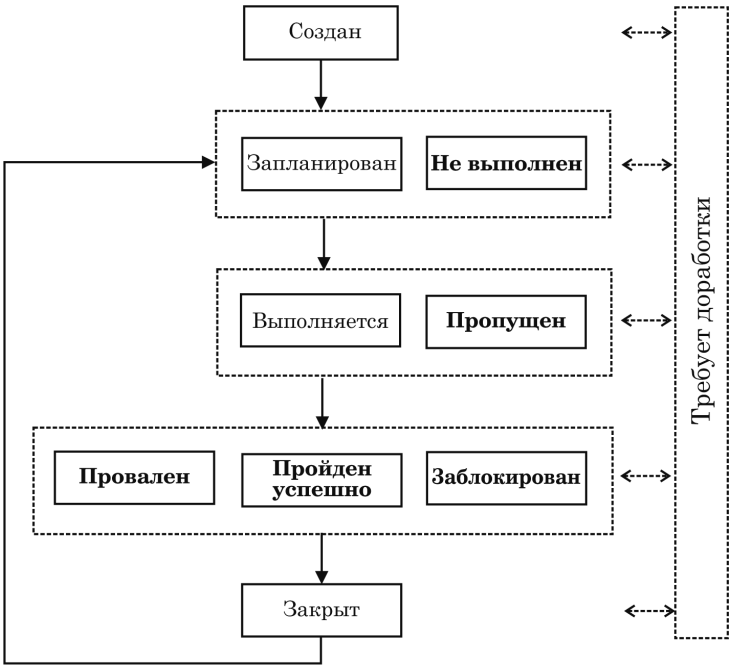

Рис. 4.1
Жизненный цикл (набор состояний) тест-кейса

Подробное описание состояний приведено в [5].

Атрибуты (поля) тест-кейса

Как уже было сказано выше, термин «тест-кейс» может относиться к формальной записи тест-кейса в виде технического документа. Эта запись имеет общепринятую структуру, компоненты которой называются атрибутами (полями) тест-кейса.

В зависимости от инструмента управления тест-кейсами внешний вид их записи может немного отличаться, могут быть добавлены или убраны отдельные поля, но концепция остаётся неизменной.

Общий вид всей структуры тест-кейса представлен на рисунке 4.2.

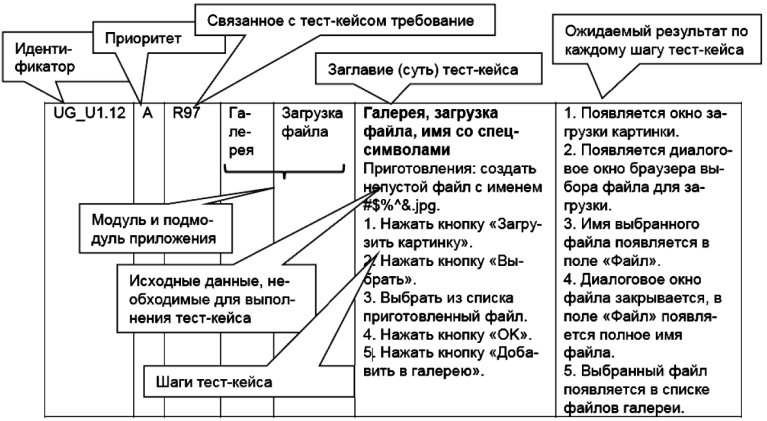

Рис. 4.2
Общий вид тест-кейса

Теперь рассмотрим каждый атрибут подробно.

Идентификатор (identifier) представляет собой уникальное значение, позволяющее однозначно отличить один тест-кейс от другого и используемое во всевозможных ссылках.

Приоритет (priority) показывает важность тест-кейса. Он может быть выражен буквами (A, B, C, D, E), цифрами (1, 2, 3, 4, 5), словами («крайне высокий», «высокий», «средний», «низкий», «крайне низкий») или иным удобным способом. Количество градаций также не фиксировано, но чаще всего лежит в диапазоне от трёх до пяти.

Приоритет тест-кейса может коррелировать с:

— важностью требования, пользовательского сценария или функции, с которыми связан тест-кейс;

— потенциальной важностью дефекта, на поиск которого направлен тест-кейс;

— степенью риска, связанного с проверяемым тест-кейсом требованием, сценарием или функцией.

Связанное с тест-кейсом требование (requirement) показывает то основное требование, проверке выполнения которого посвящён тест-кейс (основное — потому что один тест-кейс может затрагивать несколько требований). Наличие этого поля улучшает такое свойство тест-кейса, как прослеживаемость.

Модуль и подмодуль приложения (module and submodule) указывают на части приложения, к которым относится тест-кейс, и позволяют лучше понять его цель.

Модули и подмодули можно выделять на основе графического интерфейса пользователя (крупные области и элементы внутри них), на основе решаемых приложением задач и подзадач и т. д. Главное, чтобы эта логика была одинаковым образом применена ко всему приложению.

Заглавие (суть) тест-кейса (title) призвано упростить и ускорить понимание основной идеи (цели) тест-кейса без обращения к его остальным атрибутам. Именно это поле является наиболее информативным при просмотре списка тест-кейсов.

Исходные данные, необходимые для выполнения тест-кейса (precondition, preparation, initial data, setup), позволяют описать всё то, что должно быть подготовлено до начала выполнения тест-кейса.

Шаги тест-кейса (steps) описывают последовательность действий, которые необходимо реализовать в процессе выполнения тест-кейса.

Ожидаемые результаты (expected results) по каждому шагу тест-кейса описывают реакцию приложения на действия, описанные в поле «шаги тест-кейса». Номер шага соответствует номеру результата.

Практическое задание

Вы являетесь QA инженером в компании, занимающейся разработкой мобильного мессенджера. Вашей задачей является составление кейсов для дальнейшего проведения тестирования сотрудниками вашего отдела.

В качестве программного продукта, для которого вы пишете тест кейсы, можете взять любой коммерческий мессенджер (Whatsapp, ICQ, или что-то ещё).

Для определения ожидаемого поведения того или иного функционала используйте любые данные, доступные вам. В случае недостатка информации описывайте то, что, вы считаете, должно происходить.

В лабораторной работе необходимо написать чек-лист, а также тест-кейсы, объединённые в тест-сьюты (тестовые сценарии).

Примечание: в лабораторной работе ожидается написание тестов, в которых:

— предполагается тестирование «чёрного ящика»;

— по уровню тестирования, тесты — системные (или acceptance);

— написаны как позитивные, так и негативные тесты;

— используются техники тестирования классов эквивалентности и тестирования пограничных значений;

— описано тестирование пользовательских интерфейсов.

Содержание отчёта:

1. Цель работы.
2. Чек-лист.
3. Также тест-кейсы, объединённые в тестовые сценарии.
4. Выводы по работе.

Контрольные вопросы

1. Что чаще всего представляет собой чек-лист?
2. Какими свойствами должен обладать чек-лист?
3. Дайте определение тест-кейса.
4. Дайте определение тестового набора.
5. Что такое высокоуровневый тест-кейс?
6. Что такое низкоуровневый тест-кейс?
7. Что такое спецификация тест-кейса?
8. Что такое тест-сценарий?
9. Какова цель написания тест-кейсов?
10. Опишите жизненный цикл (набор состояний) тест-кейса.
10. Опишите атрибуты (поля) тест-кейса.
11. С чем может коррелировать приоритет тест-кейса?
12. Какое поле является наиболее информативным при просмотре списка тест-кейсов?

ЛАБОРАТОРНАЯ РАБОТА № 5. ПОДГОТОВКА ОТЧЁТА О ДЕФЕКТАХ

Цель: изучить принципы разработки отчёта о дефектах.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчёт и ответить на контрольные вопросы.

Теоретические сведения

Отчёт о дефекте

Отчёт о дефекте (defect report) — документ, описывающий и приоритизирующий обнаруженный дефект, а также содействующий его устранению.

Как следует из самого определения, отчёт о дефекте пишется со следующими основными целями:

— предоставить информацию о проблеме — уведомить проектную команду и иных заинтересованных лиц о наличии проблемы, описать суть проблемы;

— приоритизировать проблему — определить степень опасности проблемы для проекта и желаемые сроки её устранения;

— содействовать устранению проблемы — качественный отчёт о дефекте не только предоставляет все необходимые подробности для понимания сути случившегося, но также может содержать анализ причин возникновения проблемы и рекомендации по исправлению ситуации.

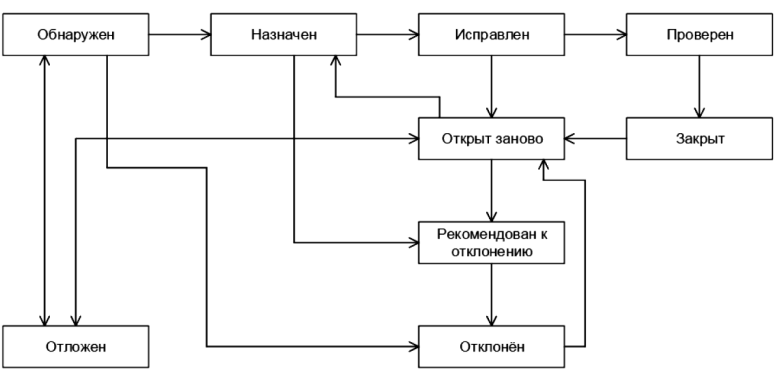

Рис. 5.1

Жизненный цикл отчёта о дефекте с наиболее типичными переходами между состояниями

Отчёт о дефекте (и сам дефект вместе с ним) проходит определённые стадии жизненного цикла, которые схематично показаны на рисунке 5.1.

Более подробно стадии жизненного цикла дефекта описаны в учебнике [5].

Атрибуты (поля) отчёта о дефекте

В зависимости от инструментального средства управления отчётами о дефектах внешний вид их записи может немного отличаться, могут быть добавлены или убраны отдельные поля, но концепция остаётся неизменной.

Общий вид всей структуры отчёта о дефекте представлен на рисунке 5.2.

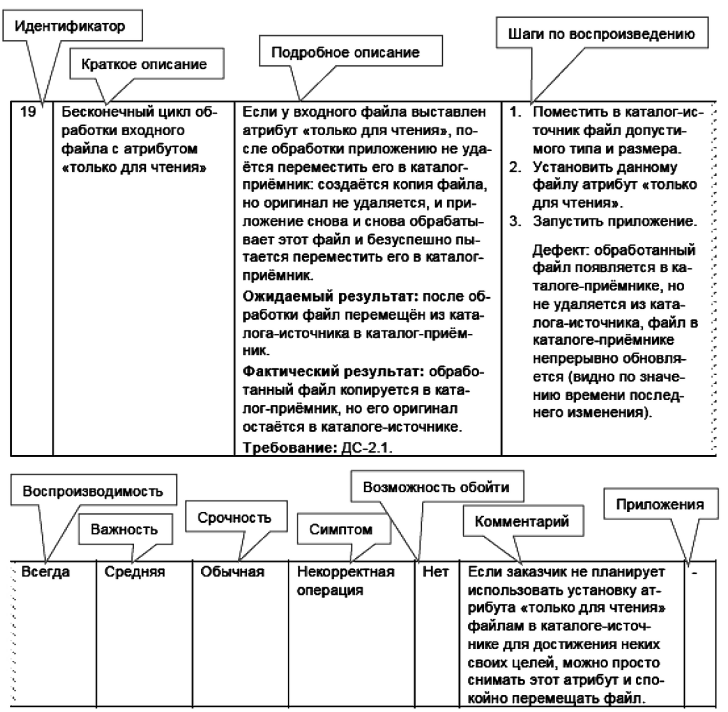

Рис. 5.2
Общий вид отчёта о дефекте

Идентификатор (identifier) представляет собой уникальное значение, позволяющее однозначно отличить один отчёт о дефекте от

другого и используемое во всевозможных ссылках. В общем случае идентификатор отчёта о дефекте может представлять собой просто уникальный номер, но (если позволяет инструментальное средство управления отчётами) может быть и куда сложнее: включать префиксы, суффиксы и иные осмысленные компоненты, позволяющие быстро определить суть дефекта и часть приложения (или требований), к которой он относится.

Краткое описание (summary) должно в предельно лаконичной форме давать исчерпывающий ответ на вопросы «Что произошло?», «Где это произошло?», «При каких условиях это произошло?». Например: «Отсутствует логотип на странице приветствия, если пользователь является администратором»:

— что произошло? Отсутствует логотип;

— где это произошло? На странице приветствия;

— при каких условиях это произошло? Если пользователь является администратором.

Одной из самых больших проблем для начинающих тестировщиков является именно заполнение поля «краткое описание», которое одновременно должно:

— содержать предельно краткую, но в то же время достаточную для понимания сути проблемы информацию о дефекте;

— отвечать на только что упомянутые вопросы («что, где и при каких условиях случилось») или как минимум на те 1–2 вопроса, которые применимы к конкретной ситуации;

— быть достаточно коротким, чтобы полностью помещаться на экране (в тех системах управления отчётами о дефектах, где конец этого поля обрезается или приводит к появлению скроллинга);

— при необходимости содержать информацию об окружении, под которым был обнаружен дефект;

— по возможности не дублировать краткие описания других дефектов (и даже не быть похожими на них), чтобы дефекты было сложно перепутать или посчитать дубликатами друг друга;

— быть законченным предложением русского или английского (или иного) языка, построенным по соответствующим правилам грамматики.

Для создания хороших кратких описаний дефектов рекомендуется пользоваться следующим алгоритмом.

1. Полноценно понять суть проблемы. До тех пор, пока у тестировщика нет чёткого, кристально чистого понимания того, «что сломалось», писать отчёт о дефекте вообще едва ли стоит.

2. Сформулировать подробное описание (description) дефекта — сначала без оглядки на длину получившегося текста.

3. Убрать из получившегося подробного описания всё лишнее, уточнить важные детали.

4. Выделить в подробном описании слова (словосочетания, фрагменты фраз), отвечающие на вопросы, «что, где и при каких условиях случилось».

5. Оформить получившееся в пункте 4 в виде законченного грамматически правильного предложения.

6. Если предложение получилось слишком длинным, переформулировать его, сократив длину (за счёт подбора синонимов, использования общепринятых аббревиатур и сокращений). К слову, в английском языке предложение почти всегда будет короче русского аналога.

Подробное описание (description) представляет в развёрнутом виде необходимую информацию о дефекте, а также (обязательно!) описание фактического результата, ожидаемого результата и ссылку на требование (если это возможно).

Шаги по воспроизведению (steps to reproduce, STR) описывают действия, которые необходимо выполнить для воспроизведения дефекта. Это поле похоже на шаги тест-кейса, за исключением одного важного отличия: здесь действия прописываются максимально подробно, с указанием конкретных вводимых значений и самых мелких деталей, так как отсутствие этой информации в сложных случаях может привести к невозможности воспроизведения дефекта.

Воспроизводимость (reproducibility) показывает, при каждом ли прохождении по шагам воспроизведения дефекта удаётся вызвать его проявление. Это поле принимает всего два значения: всегда (always) или иногда (sometimes).

Важность (severity) показывает степень ущерба, который наносится проекту существованием дефекта.

В общем случае выделяют следующие градации важности:

- критическую (critical) — существование дефекта приводит к масштабным последствиям катастрофического характера, например потери данных, раскрытию конфиденциальной информации, нарушению ключевой функциональности приложения и т. д.;
- высокую (major) — существование дефекта приносит ощутимые неудобства многим пользователям в рамках их типичной деятельности, например недоступность вставки из буфера обмена, неработоспособность общепринятых клавиатурных комбинаций, необходимость перезапуска приложения при выполнении типичных сценариев работы;

• среднюю (medium) — существование дефекта слабо влияет на типичные сценарии работы пользователей, и/или существует обходной путь достижения цели, например диалоговое окно не закрывается автоматически после нажатия кнопок «OK»/«Cancel», при распечатке нескольких документов подряд не сохраняется значение поля «Двусторонняя печать», перепутаны направления сортировок по некоему полю таблицы;

• низкую (minor) — существование дефекта редко обнаруживается незначительным процентом пользователей и (почти) не влияет на их работу, например опечатка в глубоко вложенном пункте меню настроек, некое окно сразу при отображении расположено неудобно (нужно перетянуть его в удобное место), неточно отображается время до завершения операции копирования файлов.

Срочность (priority) показывает, как быстро дефект должен быть устранён.

В общем случае выделяют следующие градации срочности:

• наивысшую (ASAP, as soon as possible) — указывает на необходимость устранить дефект настолько быстро, насколько это возможно. В зависимости от контекста «настолько быстро, насколько возможно» может варьироваться от «в ближайшем билде» до единиц минут;

• высокую (high) — означает, что дефект следует исправить вне очереди, так как его существование или уже объективно мешает работе, или начнёт создавать такие помехи в самом ближайшем будущем;

• обычную (normal) — означает, что дефект следует исправить в порядке общей очерёдности. Такое значение срочности получает большинство дефектов;

• низкую (low) — означает, что в обозримом будущем исправление данного дефекта не окажет существенного влияния на повышение качества продукта.

Симптом (symptom) позволяет классифицировать дефекты по их типичному проявлению. Не существует никакого общепринятого списка симптомов. Более того, далеко не в каждом инструментальном средстве управления отчётами о дефектах есть такое поле, а там, где оно есть, его можно настроить. В качестве примера рассмотрим следующие значения симптомов дефекта:

• косметический дефект (cosmetic flaw) — визуально заметный недостаток интерфейса, не влияющий на функциональность

приложения (например, надпись на кнопке выполнена шрифтом не той гарнитуры);

- повреждение/потеря данных (data corruption/loss) — в результате возникновения дефекта искажаются, уничтожаются (или не сохраняются) некоторые данные (например, при копировании файлов копии оказываются повреждёнными);
- проблема в документации (documentation issue) — дефект относится не к приложению, а к документации (например, отсутствует раздел руководства по эксплуатации);
- некорректная операция (incorrect operation) — некоторая операция выполняется некорректно (например, калькулятор показывает ответ 17 при умножении 2 на 3);
- проблема инсталляции (installation problem) — дефект проявляется на стадии установки и/или конфигурирования приложения;
- ошибка локализации (localization issue) — что-то в приложении не переведено или переведено неверно на выбранный язык интерфейса;
- нереализованная функциональность (missing feature) — некая функция приложения не выполняется или не может быть вызвана (например, в списке форматов для экспорта документа отсутствует несколько пунктов, которые там должны быть);
- проблема масштабируемости (scalability) — при увеличении количества доступных приложению ресурсов не происходит ожидаемого прироста производительности приложения;
- низкая производительность (low performance) — выполнение неких операций занимает недопустимо большое время;
- крах системы (system crash) — приложение прекращает работу или теряет способность выполнять свои ключевые функции (также может сопровождаться крахом операционной системы, веб-сервера и т. д.);
- неожиданное поведение (unexpected behavior) — в процессе выполнения некоторой типичной операции приложение ведёт себя необычным (отличным от общепринятого) образом (например, после добавления в список новой записи активной становится не новая запись, а первая в списке);
- недружественное поведение (unfriendly behavior) — поведение приложения создаёт пользователю неудобства в работе (например, на разных диалоговых окнах в разном порядке расположены кнопки «OK» и «Cancel»);

• расхождение с требованиями (variance from specs) — этот симптом указывают, если дефект сложно соотнести с другими симптомами, но тем не менее приложение ведёт себя не так, как описано в требованиях;

• предложение по улучшению (enhancement) — во многих инструментальных средствах управления отчётами о дефектах для этого случая есть отдельный вид отчёта, так как предложение по улучшению формально нельзя считать дефектом: приложение ведёт себя согласно требованиям, но у тестировщика есть обоснованное мнение о том, как ту или иную функциональность можно улучшить.

Возможность обойти (workaround) показывает, существует ли альтернативная последовательность действий, выполнение которой позволило бы пользователю достичь поставленной цели (например, клавиатурная комбинация Ctrl + P не работает, но распечатать документ можно, выбрав соответствующие пункты в меню). В некоторых инструментальных средствах управления отчётами о дефектах это поле может просто принимать значения «Да» и «Нет», в некоторых при выборе «Да» появляется возможность описать обходной путь. Традиционно считается, что дефектам без возможности обхода стоит повысить срочность исправления.

Комментарий (comments, additional info) может содержать любые полезные для понимания и исправления дефекта данные. Иными словами, сюда можно писать всё то, что нельзя писать в остальные поля.

Приложения (attachments) представляет собой не столько поле, сколько список прикреплённых к отчёту о дефекте приложений (копий экрана, вызывающих сбой файлов и т. д.).

Практическое задание

В ходе разработки вам как сотруднику отдела тестирования, поручили провести полное системное тестирование приложения «Калькулятор» (тестовое приложение — BugTest.exe — расположено в папке лабораторных работ).

Для данного приложения отсутствует какая-либо документация, и никто из ваших коллег или клиентов не доступен для связи. Исходя из этого, считайте любое спорное поведение багом и составляйте багрепорт.

Вам необходимо провести тестирование, описать проведённые тесты и составить отчёты о дефектах. Плюсом будет являться предоставление статистики прохождения тестов приложением.

Содержание отчёта:

1. Цель работы.
2. Описание проведённых тестов и составить отчёты о дефектах.
3. Отчёт о дефектах.
4. Выводы по работе.

Контрольные вопросы

1. Что такое отчёт о дефекте?
2. С какими основными целями пишется отчёт о дефекте?
3. Опишите жизненный цикл отчёта о дефекте.
4. Опишите атрибуты (поля) отчёта о дефекте.
5. Каким должно быть поле «краткое описание»?
6. Опишите алгоритм создания кратких описаний дефектов.
7. Назовите градации важности дефекта.
8. Назовите градации срочности устранения дефекта.
9. Назовите значения симптомов дефекта.

ЛАБОРАТОРНАЯ РАБОТА № 6. ТЕСТИРОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ

Цель: изучить принципы тестирования производительности.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчёт и ответить на контрольные вопросы.

Теоретические сведения

Тестирование производительности

Нагрузочное тестирование, или **тестирование производительности**, — это автоматизированное тестирование, имитирующее работу определённого количества бизнес-пользователей на каком-либо общем (разделяемом ими) ресурсе.

Основные виды тестирования производительности

Рассмотрим основные виды нагрузочного тестирования, также задачи, стоящие перед ними.

Тестирование производительности (Performance testing)

Задачей тестирования производительности является определение масштабируемости приложения под нагрузкой, при этом происходят:

- измерение времени выполнения выбранных операций при определённых интенсивностях выполнения этих операций;
- определение количества пользователей, одновременно работающих с приложением;
- определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций);
- исследование производительности на высоких, предельных, стрессовых нагрузках.

Стрессовое тестирование (Stress Testing)

Стрессовое тестирование позволяет проверить насколько приложение и система в целом работоспособны в условиях стресса и также оценить способность системы к регенерации, т. е. к возвращению к нормальному состоянию после прекращения воздействия стресса. Стрессом в данном контексте может быть повышение интенсивности выполнения операций до очень высоких значений или аварийное изменение конфигурации сервера. Также одной из задач при

стрессовом тестировании может быть оценка деградации производительности, таким образом цели стрессового тестирования могут пересекаться с целями тестирования производительности.

Объёмное тестирование (Volume Testing)

Задачей объемного тестирования является получение оценки производительности при увеличении объёмов данных в базе данных приложения, при этом происходит:

- измерение времени выполнения выбранных операций при определённых интенсивностях выполнения этих операций;
- может производиться определение количества пользователей, одновременно работающих с приложением.

Тестирование стабильности или надежности (Stability/Reliability Testing)

Задачей тестирования стабильности (надежности) является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки. Время выполнения операций может играть в данном виде тестирования второстепенную роль. При этом на первое место выходит отсутствие утечек памяти, перезапусков серверов под нагрузкой и другие аспекты, влияющие именно на стабильность работы.

Цели нагрузочного тестирования

Основными целями нагрузочного тестирования являются:

1. Оценка производительности и работоспособности приложения на этапе разработки и передачи в эксплуатацию.

2. Оценка производительности и работоспособности приложения на этапе выпуска новых релизов, патч-сетов.

3. Оптимизация производительности приложения, включая настройки серверов и оптимизацию кода.

4. Подбор соответствующей для данного приложения аппаратной (программной) платформы и конфигурации сервера.

Заметим, что в рамках одной цели могут использоваться разные виды тестов производительности и нагрузки, например для первой, второй и третьей целей нужно производить как тестирование производительности, так и тестирование стабильности. Но при планировании нагрузочного тестирования логичнее всё же отталкиваться от технических целей (а не коммерческих, перечисленных выше), которые достигаются в результате тестирования, и классифицировать тесты по ним:

1. Если интересует исследование производительности приложения, а именно время отклика для операций на разных нагрузках в до-

вольно широких диапазонах, включая стрессовые нагрузки, то это все-таки тестирование производительности (Performance Testing).

2. Если целью является понимание, насколько приложение устойчиво в режиме длительного использования (исключение утечек памяти, некорректных конфигурационных настроек и т. д.), то проводится долгий нагрузочный тест — это тестирование стабильности (Stability Testing). При этом анализ времен отклика может иметь место, но не быть первым приоритетом, главное, чтобы система «не упала».

3. Стресс-тестирование (Stress Testing) имеет своей целью проверку, возвращается ли система после запредельной нагрузки (и как скоро) к нормальному режиму, также целями стрессового тестирования могут быть проверки поведения системы в случаях, когда один из серверов приложения в пуле перестаёт работать, аварийно изменилась аппаратная конфигурации сервера базы данных и т. д. Отметим также, что при стрессовом тестировании проверяется не производительность системы, а её способность к регенерации после сверхнагрузки.

Практическое задание

Необходимо провести тестирование производительности веб-приложения. В качестве тестируемого приложения можете использовать любой личный веб-ресурс (работа в команде не возбраняется).

В качестве инструмента для нагрузочного тестирования можете использовать любой инструмент (например, Jmeter, Yandex Tank, Webloadui).

Вашей задачей является предоставление отчёта по следующим вопросам.

1. При каком максимальном количестве единовременных пользователей (n), 95% ответов приходят в течение 5 секунд и количество ошибок составляет меньше 0,5%?

2. Исходя из результатов тестирования производительности, какие процессы вы бы оптимизировали в первую очередь для улучшения этого результата?

3. Как поведёт себя сайт при количестве пользователей, равном $n/4$, $n/2$ и $2n$? Предоставьте перцентили времени отклика и количество ошибок.

Предоставление графиков приветствуется.

Содержание отчёта

1. Цель работы.
2. Отчёт по тестированию производительности приложения.
3. Выводы по работе.

Контрольные вопросы

1. Дайте определение нагрузочного тестирования.
2. Назовите основные виды тестирования производительности.
3. Что является задачей тестирования производительности?
4. Что является задачей стрессового тестирования?
5. Что является задачей объёмного тестирования?
6. Что является задачей тестирования стабильности (надежности)?
7. Назовите цели нагрузочного тестирования.

ЛАБОРАТОРНАЯ РАБОТА № 7. АВТОТЕСТИРОВАНИЕ

Цель: изучить принципы разработки чек-листов, тест-кейсов, наборов тест-кейсов.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчёт и ответить на контрольные вопросы.

Теоретические сведения

Автоматизированное тестирование

Автоматизированное тестирование ПО (Automation Testing) — это процесс верификации программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, выполняются автоматически при помощи инструментов для автоматизированного тестирования.

С автоматизацией тестирования, как и со многими другими узконаправленными IT-дисциплинами, связано много неверных представлений. Для того чтобы избежать неэффективного применения автоматизации, следует обходить её недостатки и максимально использовать преимущества.

Преимущества автоматизации тестирования:

- повторяемость — все написанные тесты всегда будут выполняться однообразно, то есть исключён «человеческий фактор». Тестировщик не пропустит тест по неосторожности и ничего не напутает в результатах;
- быстрое выполнение — автоматизированному скрипту не нужно сверяться с инструкциями и документациями, это сильно экономит время выполнения;
- меньшие затраты на поддержку — когда автоматические скрипты уже написаны, на их поддержку и анализ результатов требуется, как правило, меньшее время, чем на проведение того же объема тестирования вручную;
- отчёты — автоматически рассылаемые и сохраняемые отчёты о результатах тестирования;
- выполнение без вмешательства — во время выполнения тестов инженер-тестировщик может заниматься другими полезными делами, или тесты могут выполняться в нерабочее время

(этот метод предпочтительнее, так как нагрузка на локальные сети ночью снижена).

Недостатки автоматизации тестирования:

• повторяемость — все написанные тесты всегда будут выполняться однообразно. Это одновременно является и недостатком, так как тестировщик, выполняя тест вручную, может обратить внимание на некоторые детали и, проведя несколько дополнительных операций, найти дефект. Скрипт этого сделать не может;

• затраты на поддержку — несмотря на то что в случае автоматизированных тестов они меньше, чем затраты на ручное тестирование того же функционала, — они всё же есть. Чем чаще изменяется приложение, тем они выше;

• большие затраты на разработку — разработка автоматизированных тестов — это сложный процесс, так как фактически идёт разработка приложения, которое тестирует другое приложение;

• стоимость инструмента для автоматизации — в случае если используется лицензионное ПО, его стоимость может быть достаточно высока. Свободно распространяемые инструменты как правило отличаются более скромным функционалом и меньшим удобством работы;

• пропуск мелких ошибок — автоматический скрипт может пропускать мелкие ошибки, на проверку которых он не запрограммирован.

Для того чтобы принять решение о целесообразности автоматизации приложения, нужно ответить на вопрос «перевешивают ли в нашем случае преимущества?». При принятии решения стоит помнить, что альтернатива — это ручное тестирование, у которого есть свои недостатки.

Автоматизацию нужно применять, тестируя:

— труднодоступные места в системе;

— часто используемую функциональность, риски от ошибок в которой достаточно высоки;

— рутинные операции, такие как переборы данных;

— валидационные сообщения;

— длинные end-to-end сценарии;

— проверку данных, требующих точных математических расчётов;

— проверка правильности поиска данных.

Для более эффективного использования автоматизации тестирования лучше разработать отдельные тест-кейсы, проверяющие:

- базовые операции создания/чтения/изменения/удаления сущностей (так называемые CRUD операции — Create/Read/Update/Delete) (пример: создание, удаление, просмотр и изменение данных о пользователе);
- типовые сценарии использования приложения либо отдельные действия (пример: пользователь заходит на почтовый сайт, листает письма, просматривает новые, пишет и отправляет письмо, выходит с сайта. Это так называемый end-to-end сценарий, который проверяет совокупность действий. Такие сценарии позволяют вернуть систему в состояние, максимально близкое к исходному, а значит, — минимально влияющее на другие тесты);
- интерфейсы, работы с файлами и другие моменты, неудобные для тестирования вручную (пример: система создает некоторый xml файл, структуру которого необходимо проверить).

Это и есть та функциональность, от автоматизации тестирования которой можно получить наибольшую отдачу.

Рассмотрим аспекты, влияющие на выбор инструмента автоматизации тестирования.

Во-первых, необходимо обратить внимание, насколько хорошо инструмент для автоматизации распознаёт элементы управления в тестируемом приложении. В случае когда элементы не распознаются, стоит поискать плагин либо соответствующий модуль. Если такового нет — от инструмента лучше отказаться.

Во-вторых, нужно обратить внимание на то, сколько времени требуется на поддержку скриптов, написанных с помощью выбранного инструмента.

И последний момент, на который нужно обратить внимание, — насколько удобен инструмент для написания новых скриптов. Сколько требуется на это времени, насколько можно структурировать код, насколько код читаем, насколько удобна среда разработки для рефакторинга и т. п.

Три уровня автоматизации тестирования

Условно, тестируемое приложение можно разбить на три уровня:

- уровень модульного тестирования **(**Unit Tests Layer);
- уровень функционального тестирования (Functional Tests Layer (Non-UI));
- уровень тестирования через пользовательский интерфейс (GUI Test Layer).

Для обеспечения лучшего качества продукта рекомендуется автоматизировать все три уровня. Рассмотрим более детально стратегию автоматизации тестирования на основе трёхуровневой модели.

Под автоматизированными тестами на уровне модульного тестирования понимаются Компонентные, или Модульные, тесты, написанные разработчиками. Наличие подобных тестов на ранних стадиях проекта, а также постоянное их пополнение новыми тестами, проверяющими «баг фиксы», убережёт проект от многих серьёзных проблем.

Как правило, не всю бизнес-логику приложения можно протестировать через GUI слой. Это может быть особенностью реализации, которая прячет бизнес-логику от пользователей. Именно по этой причине по договорённости с разработчиками для команды тестирования может быть реализован доступ напрямую к функциональному слою, дающий возможность тестировать непосредственно бизнес логику приложения, минуя пользовательский интерфейс.

На уровне тестирования через пользовательский интерфейс есть возможность тестировать не только интерфейс пользователя, но также и функциональность, выполняя операции вызывающую бизнес-логику приложения. С нашей точки зрения, такого рода сквозные тесты дают больший эффект, нежели просто тестирование функционального слоя, так как мы тестируем функциональность, эмулируя действия конечного пользователя через графический интерфейс.

Практическое задание

Вы — инженер по контролю качества, занимающийся тестированием интернет-магазина.

Вашей задачей является написание и автоматизация позитивных и негативных проверок по следующим сценариям:

— регистрация;
— авторизация;
— навигация по основным разделам магазина;
— поиск товара;
— фильтрация отображаемых товаров;
— добавление товара в корзину;
— редактирование корзины;
— (на тестовых сайтах) совершение покупки.

В качестве тестового интернет-магазина можете использовать http://automationpractice.com.

Автотесты необходимо писать в Selenium Webdriver.

Использование паттерна Page Object в автотестах — обязательно.

Содержание отчёта:

1. Цель работы.
2. Коды автотестов
3. Отчёт по прохождению автотестов.
4. Выводы по работе.

Контрольные вопросы

1. Дайте определение автоматизированного тестирования ПО.
2. Охарактеризуйте преимущества автоматизации тестирования.
3. Охарактеризуйте недостатки автоматизации тестирования.
4. Для каких задач нужно применять автоматизированное тестирование?
5. Какие отдельные тест-кейсы нужно разработать для эффективного использования автоматизации тестирования?
6. Какие аспекты влияют на выбор инструмента автоматизации тестирования?
7. Охарактеризуйте три уровня автоматизации тестирования.

ЛАБОРАТОРНАЯ РАБОТА № 8. ИССЛЕДОВАТЕЛЬСКОЕ ТЕСТИРОВАНИЕ

Цель: изучить принципы исследовательского тестирования.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчёт и ответить на контрольные вопросы.

Теоретические сведения

Исследовательское тестирование

Простейшее определение исследовательского тестирования — это разработка и выполнение тестов в одно и то же время, что является противоположностью сценарного подхода (с его предопределёнными процедурами тестирования, неважно ручными или автоматизированными). Исследовательские тесты, в отличие от сценарных тестов, не определены заранее и не выполняются в точном соответствии с планом.

Исследовательское тестирование является мощным и приятным подходом к тестированию. В некоторых случаях оно может быть более продуктивным, чем привычное тестирование по сценариям. Кроме очевидной проблемы с тестированием на основе тест-кейсов, состоящей в высоких затратах времени, существует ещё одна — существующие техники оптимизации направлены на то, чтобы максимально исследовать приложение во всех учтённых ситуациях, которые мы можем контролировать, — но невозможно учесть и проконтролировать всё. Исследовательское же тестирование часто позволяет обнаружить дефекты, вызванные этими неучтёнными факторами. К тому же оно прекрасно показывает себя в следующих ситуациях: отсутствия или низкого качества необходимой документации; необходимости быстрой оценки качества при нехватке времени; подозрении на неэффективность имеющихся тест-кейсов; необходимости проверить компоненты, разработанные «третьими сторонами»; верификации устранения дефекта (для проверки, что он не проявляется при незначительном отступлении от шагов воспроизведения).

Результаты исследовательского тестирования не обязательно радикально отличаются от тех, которые мы получаем с помощью сценарного тестирования, и оба этих подхода к тестированию являются полностью совместимыми. Такие компании, как Nortel и Microsoft, обычно используют оба подхода в одном проекте. Тем не менее есть много важных различий между двумя подходами.

Как и перед любым другим видом тестирования, предварительно необходимо получить базовое понимание требований к программному продукту (в крайнем случае — сильно поможет даже словесное описание разработчиками желаемого поведения) и обозначить область тестирования. Чаще всего исследовательскому тестированию подвергается новый функционал, который необходимо срочно протестировать и выпустить в релиз, тем самым ограничивая область, подвергаемую тестированию.

Процесс исследовательского тестирования обычно выглядит следующим образом.

1. Происходит базовое ознакомление с функционалом и определяется область тестирования.

2. Создаётся чек-лист с базовым описанием тестов, покрывающих область тестирования.

3. Начинается процесс тестирования, который основывается на проверках из чек-листа. При этом в ходе тестирования и улучшения понимания ПО тестировщикам могут прийти в голову проверки, не описанные в изначальном чек-листе, и в таких случаях чек-лист стоит обновлять по ходу тестирования.

4. В ходе тестирования формируются отчёты об ошибках.

5. Предоставляется отчёт о проведении тестирования. Для этих целей может использоваться обновлённый чек-лист, показывающий осуществлённые проверки и результаты этих проверок.

Практическое задание

Вам необходимо протестировать произвольный программный продукт, к разработке которого вы не имеете никакого отношения. Ограничьте области, подвергаемые тестированию, в зависимости от сложности выбранного ПО (ожидается, что в ходе тестирования будет осуществлено около 50 различных проверок).

Содержание отчёта:

1. Цель работы.
2. Описание общих требований к ПО.
3. Баг-репорты по найденным ошибкам.
4. Общий отчёт о проведении тестирования.
5. Выводы по работе.

Контрольные вопросы

1. Дайте определение исследовательского тестирования.

2. В каких ситуациях нужно использовать исследовательское тестирование?

3. Должны ли радикально отличаться результаты исследовательского тестирования от тех, которые получены с помощью сценарного тестирования?

4. Совместимы ли исследовательское и сценарное тестирование?

5. Что чаще всего подвергается исследовательскому тестированию?

6. Как обычно выглядит процесс исследовательского тестирования?

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. ISTQB Стандартный глоссарий терминов, используемых в тестировании программного обеспечения. — 2014. — 73 с.

2. ГОСТ Р ИСО/МЭК 12119-2000. Информационная технология. Пакеты программ. Требования к качеству и тестирование.

3. ГОСТ Р ИСО/МЭК 12207-99. Информационная технология. Процессы жизненного цикла программных средств.

4. *Вигерс, К.* Разработка требований к программному обеспечению / пер. с англ. — 3-е изд., дополненное. — М. : Русская редакция ; СПб. : БХВ-Петербург, 2014. — 736 с.

5. *Куликов, С. С.* Тестирование программного обеспечения. Базовый курс : практ. пособие. — Минск : Четыре четверти, 2015. — 294 с.

6. *Липаев, В. В.* Тестирование компонентов и комплексов программ : учебник. — М. : СИНТЕГ, 2010. — 400 с.

7. *Майерс, Г.* Искусство тестирования программ / Г. Майерс, Т. Баджетт, К. Сандлер. — М. : ДИАЛЕКТИКА, 2016. — 272 с.

8. *Пышкин, Е. В.* Модульное тестирование программного обеспечения. Профессиональный базовый курс с практикой на JUnit / Е. В. Пышкин, М. И. Глухих ; под ред. М. В. Финкова. — СПб. : Профессиональная литература : АйТи-Подготовка, 2015. — 239 с.

9. *Савин, Р.* Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах. — М. : Дело, 2007. — 312 с.

СОДЕРЖАНИЕ

Александр Владимирович ИГНАТЬЕВ

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Учебное пособие

Издание третье, стереотипное

Зав. редакцией литературы
по информационным технологиям
и системам связи *О. Е. Гайнутдинова*

ЛР № 065466 от 21.10.97
Гигиенический сертификат 78.01.10.953.П.1028
от 14.04.2016 г., выдан ЦГСЭН в СПб

Издательство «ЛАНЬ»
lan@lanbook.ru; www.lanbook.com;
196105, Санкт-Петербург, пр. Юрия Гагарина, 1, лит. А.
Тел.: (812) 412-92-72, 336-25-09.
Бесплатный звонок по России: 8-800-700-40-71

Подписано в печать 14.10.22.
Бумага офсетная. Гарнитура Школьная. Формат 84×108 $^{1}/_{32}$.
Печать офсетная/цифровая. Усл. п. л. 2,94. Тираж 100 экз.

Заказ № 1394-22.

Отпечатано в полном соответствии
с качеством предоставленного оригинал-макета
в АО «Т8 Издательские Технологии».
109316, г. Москва, Волгоградский пр., д. 42, к. 5.