Министерство образования Республики Беларусь
Учреждение образования «Полоцкий государственный университет»

Факультет информационных технологий
Кафедра технологий программирования

Методические указания
к выполнению лабораторной работы №10

по дисциплине «**Надёжность программного обеспечения**»
для специальности 1-40 01 01 Программное обеспечение информационных технологий

на тему «**Проектирование и разработка тестов**»

Новополоцк, 2017 г.

Название: «Проектирование и разработка тестов».

Цель работы: ознакомиться с основами проектирования и разработки тестов. Научиться проектировать тесты и разрабатывать тестовые случаи.

Теоретическая часть

Ключ к успешным испытаниям лежит в эффективности выбранных тестовых случаев. Четкое планирование и подготовка тестирования играют большую роль, но в условиях проведения окончательного анализа, выбранные случаи тестирования должны обеспечить обнаружение дефектов в программном продукте, иначе все планирование и подготовка окажутся напрасными.

Диаграмма действий, обеспечивающих проектирование и разработку тестов, показана на рисунке 1. Основными входными данными для этого процесса является набор документов, составляющих план проведения испытаний. План проведения испытаний должен дать описание подхода, который предусматривается задействовать при проведении тестирования, а также объемы трудозатрат на тестирование. В нем должна быть определена архитектура тестов, т.е., по меньшей мере, должны быть определены соответствующие наборы тестов. В нем также должен быть определен набор конфигураций средств тестирования, вокруг которых проектируются тесты.

Выходными результатами таких видов деятельности, как проектирование и разработка, являются набор пересмотренных и отлаженных тестовых случаев, которые готовы для использования в системных и приемочных испытаниях. Должно быть обеспечено обратное отображение тестовых случаев на технические требования. Кроме того, тестовые случаи должны обеспечить хорошее покрытие тестами, по меньшей мере, всех технических требований с наивысшими приоритетами, а в идеальном случае – абсолютно всех требований заказчика. Тестовые случаи должны обеспечить хорошее покрытие программного кода продукта за счет выполнения большей части, если не всех, логических путей в программном коде. По мере возможностей, существенная часть тестовых случаев должна быть автоматизирована для поддержки высококачественно регрессивного тестирования.

Рисунок 1 – Проектирование и реализация тестов

Как показано на рисунке 1, с разработкой тестовых случаев связан собственный жизненный цикл. Цикл предусматривает стадию проектирования, на которой формулируются цели теста, спецификации входных данных, определяются конфигурации средств тестирования. Цикл включает также этап разработки, в рамках которого даются подробные определения методик тестирования, а также этап проверки и отладки, на котором тесты подвергаются пересмотру и отладке.

Разработка тестов

Подготовка к тестированию подобна разделению луковицы на чешуйки – это многоуровневый процесс последовательной конкретизации условий. Подготовка начинается с концептуального определения стратегии тестирования, затем к нему добавляются все большее количество слоев детализации, которые описывают архитектуру тестирования и условия испытаний. В конечном итоге будут разработаны проверенные и отлаженные методики тестирования, собрана испытательная система, после чего можно смело приступать к системным испытаниям.

Разработку тестовых случаев можно сравнить со снятием одного слоя луковицы.

Проекты тестов содержат в себе больше подробностей, нежели концептуальный план построения тестов, но в то же время они еще не предусматривают конкретных действий, необходимых для прогона конкретного теста. Многоуровневый подход к разработке тестов подобен подходу, используемому при разработке программного обеспечения. В рамках высокотехнологичного процесса разработчики не переходят от формулирования требований непосредственно к написанию программных кодов. Сначала они выполняют предварительный или системный проект, в котором определяют концепцию программного продукта, после чего

составляют программу или рабочий план, в котором оговариваются все детали проектирования. В области тестирования план проведения испытаний играет такую же роль, что и проектное задание в области разработки программного обеспечения, а проект теста соответствует рабочему проекту программного обеспечения.

Одним из признаков высокого качества технического проектирования является его *модульность*. В условиях модульного проектирования система разбивается на отдельные компоненты, при этом каждый компонент имеет свое назначение и четко определенные входы и выходы. Принципы модульного проектирования часто применяются при проектировании программного обеспечения; они так же хорошо подходят и для проектирования тестов. Модульным компонентом в тестировании является тестовый случай. Это означает, что перед каждым тестовым случаем должна быть поставлена четко сформулированная цель, чтобы было ясно, что подвергается тестированию. Для каждого тестового случая должна быть строго определена среда тестирования с известными начальными условиями, благодаря чему можно рассчитывать на то, что при каждом прогоне конкретный тест будет выдавать одни и те же результаты. Наконец, каждый тестовый случай должен давать строго определенный ожидаемый результат (выход) с тем, чтобы можно было воспользоваться однозначным критерием успешного/неудачного испытания.

Другим свойством модульного проектирования является то, что модули организуются в иерархию. Иерархия есть результат разбиения системы на компоненты, и она позволяет в каждый конкретный момент времени работать с одним уровнем системы. Это обстоятельство отражается на архитектуре тестов, которая может включать в себя несколько тестовых наборов, ориентированных на проверку высокоуровневых функциональных свойств системы, а также тестовые наборы или тестовых случаи, служащих для проверки детализированных функциональных свойств системы. Например, может быть один тестовый набор, предназначенный для проверки GUI интерфейса, и еще один тест для проверки средств обеспечения безопасности системы. В состав тестового набора могут быть включены, с одной стороны, тесты, которые работают с экранами ввода специфических данных или с экранами для определения запросов. Тестовый набор отображается на одно конкретное требование или на некоторый набор логически связанных требований, в то время как более детализированные тестовые случаи отображаются на элементы функциональной спецификации системы.

И последнее, что необходимо сказать о модульности, это то, что она способствует обнаружению дефектов на ранних стадиях цикла разработки. Теоретически каждый из уровней, на которые разбита система, должен проверяться на предмет присутствия дефектов, а только после этого становится возможным переход на следующий уровень. На практике план проведения испытаний может пересматриваться и корректироваться перед переходом на следующий уровень, коим является проектирование тестов.

Проекты тестов могут пересматриваться и корректироваться до того, как начнется работа над детализированными тестовыми случаями. Эти промежуточные контрольные точки препятствуют распространению дефектов рабочих вариантов тестов далее по жизненному циклу тестирования, что может служить причиной возникновения проблем, приводящих к срывам рабочего графика.

Документ проектов тестов

Назначение документа проектов тестов состоит в том, чтобы собрать в одном месте всю информацию, порожденную в результате различных видов тестовой деятельности. Документ проектов тестов может быть представлен в виде электронной таблицы, в виде таблицы текстового процессора или в виде базы данных. Если вы пользуетесь инструментальным средством автоматизированного управления требованиями, его можно применить для сбора информации о проекте теста. Пример записи в документе проектов тестов показан в таблице 1.

Таблица 1 – Пример записи в проектном документе тестов

Идентификатор определения требования	**Идентификатор системного случая тестирования**	**Входные данные теста**	**Конфигурация теста**	**Назначение теста**
RD2.2.1	ST2.2.4	TP_lnput1.doc	TC2.0	T02.2.4. Проверить, что квалифицированный пользователь может отыскать и ознакомиться с любым ранее созданным документом, содержащим план проведения испытаний.

Эта таблица в какой-то степени похожа на матрицу RTM (Requirement Traceability matrix — матрица прослеживаемости требований). Два первых столбца таблицы должны соответствовать вхождениям в матрицу RTM, остальные столбцы соответствуют данным.

Как только проектный документ тестов будет готов, его нужно проверить в контексте связанных с ним материалов: документа определения требований (технического задания), определения входных данных теста и конфигурации средств тестирования. Цели такой проверки связаны с необходимостью убедиться:

– что рассматриваемым документом проектов тестов были охвачены все технические требования. Если то или иное требование не тестируется, в матрице RTM или в документе проектов тестов должна присутствовать запись, поясняющая, почему соответствующее требование не покрывается тестом;

– что каждый тестовый случай поддерживается соответствующими входными данными;

– что для каждого тестового случая имеется подходящая конфигурация, причем эти конфигурации не являются избыточными.

Документ проектов тестов служит основой для разработки детальной методики тестирования. В результате его пересмотра тесты могут распределяться среди исполнителей группы тестирования для дальнейшей детализации.

Разработка тестовых случаев

Тестовый случай представляет собой основной компонент динамического тестирования. По сути дела, этап системного тестирования — едва ли что-то большее, нежели просто прогон тестовых случаев на некоторой последовательности программных сборок с целью обнаружения и устранения дефектов. Это означает, что основная обязанность специалиста по тестированию заключается в написании и прогоне тестовых случаев.

Тест (test) представляет собой набор операций, предназначенных для получения одного или большего числа ожидаемых результатов в некоторой программной системе. Если получены все ожидаемые результаты, считается, что тест прошел (т.е. выполнен успешно). Если фактический результат отличается от ожидаемого, считается, что тест не прошел (т.е. завершился неудачно).

Первое, что следует отметить в приведенном определении, так это то, что каждый тест состоит из двух компонентов: (1) совокупность выполняемых вами действий и (2) последовательность событий, которые должны произойти в результате этих действий. Выполняемые действия суть тестовые действия, которые в совокупности образуют методику тестирования. Последовательность событий, происходящих в результате этих действий, называются ожидаемыми результатами. Чтобы тест был эффективным, должны быть четко и однозначно определены как методика, так и ожидаемые результаты.

Во-вторых, если методика тестирования и ожидаемые результаты определены правильно, тест должен давать результат, по которому можно сделать однозначный вывод относительно успеха или неудачи испытания. При вводе в программу двух чисел с целью получения их суммы тест считается пройденным, если на выходе программы будет получен корректный результат; в противном случае тест рассматривается как не пройденный.

Для удобства выполнения тесты можно и далее разбивать на тестовые случаи. Если некоторый тест требует выполнения пространной методики тестирования с множеством ожидаемых результатов, имеет смысл разбить такой тест на тестовые случаи. Однако при этом следует иметь в виду, что тестовый случай есть наименьший модуль тестирования, и что с каждым тестовым случаем должен быть связан, по меньшей мере, один ожидаемый результат.

В силу того, что целью тестирования является выявление дефектов, хорошим тестом считается тест с высокой вероятностью обнаружения дефекта. Чтобы спроектировать тест, обладающий высокой вероятностью обнаружения дефекта, специалист по тестированию должен стать на позицию «конструктивного разрушения» программного продукта. «Конструктивное разрушение» означает выявление проблем в программном продукте с целью их устранения. Сложность при этом связана с тем, что в задачи специалиста по тестированию не входит выявление обстоятельств, при которых программный продукт работает; наоборот, тестировщик пытается обнаружить такие ситуации, при которых продукт *не* работает. При проектировании тестового случая важно не делать никаких предположений относительно того, что та или иная функция программного продукта работает исправно. Вы не должны рассчитывать на то, что какая-либо сборка программного кода будет установлена правильно, или на то, что структура базы данных соответствует проекту, или что программа самостоятельно восстановится после потери напряжения в электросети. Нельзя также рассчитывать и на то, что если программа успешно работает на заданной платформе, то она будет работать столь же хорошо и на компьютере с меньшим пространством памяти или под управлением другой операционной системы.

Другое свойство хорошо спроектированного теста – его повторяемость. Если прогон теста завершился неудачей, очень важно воспроизвести точные условия, при которых это произошло. По этой причине важно, чтобы начальное состояние системы было определено тестовым случаем в терминах используемой версии программного продукта, конфигурации аппаратных средств, количества пользователей системы и т.д. Важно также, чтобы была известной точная методика, используемая при выявлении неисправностей. В идеальном случае должна фиксироваться точная последовательность нажатых клавиш, щелчков мыши или другие события, которые привели к отмеченным отклонениям от ожидаемых результатов. На практике все эти подробности могут оставаться неизвестными, возможно, из-за того, что методика тестирования не расписана во всех подробностях, или из-за того, что произошло какое-то неконтролируемое случайное событие без ведома тестировщиков. Вот почему, когда происходит сбой, важно, не жалея времени, немедленно воспроизвести неисправность, подробно фиксируя при этом действия, которые привели к сбою, если только они явно не расписаны в методике испытаний.

Хорошо спроектированный тест снабжен одним или большим числом четко определенных ожидаемых результатов и четко определенными критериями успешных/неудачных испытаний. Обычно ожидаемые результаты и критерии успешных/неудачных испытаний тесно связаны. Если ожидаемый результат или некоторый набор ожидаемых результатов будет получен, когда система переводится из одного заданного состояния в другое, то тестовый случай проходит. Если ожидаемые результаты не удается

зафиксировать после выполнения заданных действий, тест не проходит. Например, предположим, что в поле ввода данных вводится значение почтового индекса, и при этом ожидается, что на экран будет выведено название соответствующего штата после щелчка на кнопке «Укажите штат». Если после ввода некоторого набора почтовых индексов для каждого из них появляется правильное название штата, это значит, что тест получает ожидаемые результаты и поэтому проходит. Если же при вводе почтового кода Сиэтла на экране отображается штат Флорида, считайте, что вам повезло в плане обнаружения дефекта, поскольку ожидаемый результат не был получен.

Следует отметить, по меньшей мере, один дополнительный признак хорошо спроектированного тестового случая – он не должен быть избыточным. Оптимальный рабочий график тестирования требует, чтобы вы выполнили объем тестирования, достаточный для обнаружения всех неисправностей перед поставкой программного продукта заказчику, но вы не должны тратить время на прогон избыточных тестов. В рассмотренном выше примере вы не должны тратить время на отображение на экране названия штата для каждого известного почтового индекса, если эта функция не критична для бизнес-операций заказчика. Если с отображением на экране почтовых индексов связана крупная неприятность, наверняка потребуется потратить время на тестирование функции отображения. Кроме того, следует задуматься над вопросами автоматизации этой утомительной задачи. Но если это свойство реализуется из соображений удобства, возможно, потребуется лишь проверить какой-нибудь допустимый и несколько недопустимых индексов, дабы подтвердить правильность базовых функциональных свойств.

Резюмируя, можно утверждать, что с тестовым случаем должны быть связаны следующие признаки:

– он должен обладать высокой вероятностью обнаружения дефекта;

– он должен быть воспроизводимым;

– он должен обладать четко определенными ожидаемыми результатами и критериями успешного или неудачного выполнения теста;

– он не должен быть избыточным.

Разработка детализированных методик тестирования

Детализированные методики тестирования могут быть разработаны на основе проектов тестов. Уровень детализации методик тестирования, представленных в письменном виде, зависит от квалификации и знаний исполнителей, которые выполняют прогон тестов. Вы планируете привлечь к выполнению системного испытания программного продукта исполнителей, мало знакомых с этим продуктом? Если это так, то они должны пройти определенную подготовку, а методика тестирования должна быть четко сформулирована с тем, чтобы они выполняли корректные тестовые действия. В случае, когда тестировщики обладают подробными знаниями программного продукта, методика тестирования может быть менее детализированной. Решение относительно того, каким должен быть уровень

детализации методик тестирования, имеет большое значение, ибо можно напрасно потратить время на описание излишних подробностей для подготовленного пользователя. С другой стороны, напрасно потратить время можно и на обучение неподготовленных тестировщиков тому, как проводить испытания, если методика не содержит необходимых деталей. Так или иначе, желательно найти разумный компромисс.

Отдельного рассмотрения заслуживает один специальный случай. Если тест заслуживает того, чтобы быть автоматизированным, целесообразно выделить время на предварительную разработку детализированной методики тестирования, с помощью которой инженер по автоматизации сможет однозначно сформулировать задачу автоматизации. Расплывчатая методика тестирования, скорее всего, приведет к появлению неточных или неэффективных автоматизированных тестов. Рекомендуемый уровень детализации методики тестирования прояснится после дальнейшего обсуждения ожидаемых результатов.

Существует широкий выбор технологий, которые могут использоваться для тестирования программных продуктов. В этой главе мы будем рассматривать технологии тестирования методом черного ящика. Тестирование методом черного ящика представляет собой тестирование на системном уровне, которое имеет дело только с «внешними» аспектами программы. Тестирование методом черного ящика не предполагает каких-либо знаний о внутреннем функционировании программного продукта и проводится с использованием только внешних интерфейсов, таких как пользовательские интерфейсы или интерфейсы API (Application Programming Interface – интерфейс программирования приложений).

Двумя широко распространенными технологиями разработки тестов для тестирования методом черного ящика являются разбиение на классы эквивалентности и анализ граничных значений. Обе технологии помогают уменьшить общее количество тестовых случаев или проверяемых условий, которые необходимы для полного покрытия функциональных возможностей программы. Вполне понятно, что эти методы являются ценной частью понятия быстрого тестирования, ибо чем меньше требуется разрабатывать и прогонять тестов с целью получения того же функционального покрытия, тем эффективнее становится тестирование.

Разбиение на классы эквивалентности. Разбиение на классы эквивалентности представляет собой технологию проектирования тестов, ориентированную на снижение общего числа тестов, необходимых для подтверждения корректности функциональных возможностей программы. Основная идея, стоящая за разбиением на классы эквивалентности, заключается в том, чтобы разбить область ввода программы на классы данных. Если проектировать тесты для каждого класса данных, но не для каждого члена класса, то общее количество требуемых тестов уменьшается.

В качестве примера рассмотрим программу отображения почтовых индексов, которая рассматривалась ранее в главе. Предположим, что когда пользователь вводит пятизначный почтовый индекс и общую массу

отправляемого груза в унциях, программа возвращает стоимость доставки пакета. Областью ввода для этой программы являются почтовые индексы и масса брутто отправляемого груза. Область ввода почтового кода может быть разбита на класс допустимых вводов и класс недопустимых вводов следующим образом:

– допустимыми вводами являются все пятизначные наборы цифровых символов, образующих рабочий почтовый код;

– недопустимыми вводами являются:

– наборы цифровых символов, содержащие менее пяти символов;

– наборы цифровых символов, содержащие более пяти символов;

– наборы из пяти символов, не являющиеся рабочим почтовым кодом;

– наборы из нецифровых символов.

Аналогичное разбиение может быть построено и для массы брутто отправляемого груза. Например, требуется программа, работающая только с грузами, масса которых находится в диапазоне от 1 до 100 унций. В этом случае числовые значения, попадающие в диапазон от 1 до 100, включая и конечные точки, являются допустимыми. Все значения меньше 1 и больше 100, отрицательные значения и нецифровые значения образуют недопустимые классы ввода.

Нужно спроектировать такие тесты, которые выполняют проверку, по меньшей мере, одного представителя каждого допустимого класса ввода и, по меньшей мере, одного представителя недопустимого класса ввода. В результате тестирования с применением допустимых вводов должны быть получены однозначно определенные и ожидаемые результаты. Для почтового кода 78723 и массы в 20 унций ожидается получить конкретное значение стоимости, и это значение должно быть определено в функциональных требованиях. В случае ввода недопустимых данных должно быть получено соответствующее сообщение об ошибке, если оно определено в технических требованиях и спецификациях. При вводе недопустимых данных программа, по меньшей мере, не должна завершаться аварийно, вызывать искажение данных или вести себя непредсказуемым образом.

Анализ граничных значений. Анализ граничных значений представляет собой технологию проектирования тестов, которая является дополнением разбиения на классы -эквивалентности. Вместо того чтобы выбирать некоторый конкретный элемент класса эквивалентности, анализ граничных значений предлагает проектировщику теста выбрать элементы, которые находятся «на границе» класса. Экспериментально было доказано, что дефекты имеют тенденцию концентрироваться на границе области ввода, а не в ее центре. Не особенно ясно, почему так получается, это всего лишь установленный факт.

Например, в случае программы, которая для почтового индекса и веса отправляемого груза вычисляет стоимость доставки, анализ граничных

значений позволяет применять в качестве тестовых значений минимальное и максимальное значение веса (1 унция и 100 унций), а также ближайшее значение, меньшее минимально допустимого (0 унций), и ближайшее значение, большее максимально допустимого (101 унция). Эти значения позволяют проверить границы диапазона допустимых значений, а также значения, выходящие за пределы этого диапазона.

Определение ожидаемых результатов

Основу динамического тестирования составляет прогон управляемого набора операций на конкретной сборке программного продукта и сравнение полученных результатов с ожидаемыми результатами. Если получены ожидаемые результаты, то тест считается прошедшим на этой сборке; если зафиксировано аномальное поведение, то считается, что тест на этой сборке не прошел, в то же время этот тест, возможно, позволил обнаружить очередную неисправность. Непременным условием для определения, прошел или не прошел конкретный тест, является однозначное определение ожидаемых результатов этого теста.

Таблица 2 – Примеры ожидаемых результатов

Шаг	**Действие**	**Ожидаемый результат**	**Отметка** (V)
1	Щелкнуть на элементе «Стоимость Доставки» в главном меню.	На экран выводится меню Стоимость доставки.	
2	Ввести значение «101» в поле веса доставляемого груза.	Сообщение об ошибке «Неправильно указан вес доставляемого груза».	
3	Ввести значение «0» в поле веса доставляемого груза.	Сообщение об ошибке «Неправильно указан вес доставляемого груза».	
4	Ввести значение «100» в поле веса доставляемого груза.	На экран выводится «100 унций» как вес доставляемого груза	

Фрагмент тестового случая для программы, которую рассматривается в качестве примера и вычисляет стоимость доставки по почтовому индексу груз с заданным весом, приводится в таблице 2. Шаги методики тестирования пронумерованы в первом столбце. Краткие описания действий, выполняемых тестировщиком, даны во втором столбце, а исчерпывающие описания ожидаемых результатов находятся в третьем столбце. Предполагается, что в этом примере при проведении испытаний будет использоваться твердая (или интерактивная) копия этой таблицы, в связи с чем в ней предусмотрен четвертый столбец, в котором тестировщик может отмечать каждый выполненный шаг. Особое внимание следует обратить на

то, чтобы на каждом шаге методики тестирования предоставлялись четкие описания ожидаемых результатов.

Шаблон тестового случая

Все предложения, касающиеся проектирования тестового случая, обсуждавшиеся до сих пор, могут быть сведены в единый шаблон тестового случая. Возможно, возникнет желание создать собственный шаблон, тем не менее, стоит обратить внимание на один из вариантов такого шаблона, показанный на рисунке 2. Тестовые случаи, основанные на этом шаблоне, можно распечатать для целей неавтоматизированного тестирования, либо за счет организации отображения в браузере их HTML-версий появится возможность прогона тестов в интерактивном режиме.

Информация о тестовом случае			
Идентификатор тестового случая	SC03 ver3.0		
Владелец теста	Джин Дуглас (Jean Douglas)		
Местонахождение теста (путь)	TestServer:D:\TestProject\TestSuite\SC03.doc		
Дата последнего пересмотра	Месяц/день/год		
Тестируемое требование	SC101		
Конфигурация средств тестирования	ST02		
Взаимозависимость тестовых случаев	Выполнить прогон теста SC01 и его настройку перед прогоном данного теста.		
Цель теста	Проверить, что допустимые входные значения веса отправляемого груза дают правильные значения стоимости доставки, и что недопустимые входные значения приводят к выдаче сообщений об ошибке.		
Методика тестирования			
Настройка на прогон теста	Не проводится		N/A
Шаг	**Действие**	**Ожидаемый результат**	**Отметка (V)**
1.	Щелкнуть на элементе "Стоимость доставки" в главном меню.	Отображается меню "Стоимость доставки"	(V)
2.	Ввести "101" в поле веса доставляемого груза	Сообщение об ошибке "Неправильно указан вес доставляемого груза"	(V)
3.	Ввести "0" в поле веса доставляемого груза	Сообщение об ошибке "Неправильно указан вес доставляемого груза"	X
4.	Ввести "100" в поле веса доставляемого груза	Указан вес доставляемого груза "100 унций"	(V)
5.	Ввести "1" в поле веса доставляемого груза	Указан вес доставляемого груза "1 унция"	(V)
Очистка после прогона теста	Не производится		N/A
Результаты теста			
Тестировщик: JD	**Дата прогона теста:** месяц/день/год	**Результат теста (P/F/B): F**	
Примечания: - Тест потерпел неудачу на шаге 3. - При возникновении неисправности выдается код ошибки BR1011.			

Рисунок 2 – Пример тестового случая

Основными элементами шаблона тестового случая являются:

– идентификатор тестового случая – включает номер версии теста;

– владелец теста – имя или инициалы лица, эксплуатирующего тест (оно может не совпадать с именем автора теста);

– дата последнего пересмотра – эта информация позволит определить, является ли тест актуальным;

– наименование теста – описательное имя теста, которое позволяет легко отыскать тест и понять его назначение. Применение имен, не несущих смысловой нагрузки, например, «xxxLLL0123.tst», не рекомендуется;

– местонахождение теста – это полное имя пути, включая сервер;

– тестируемое техническое требование – это должен быть уникальный идентификатор, который отображается на документы с техническими требованиями;

– цель тестирования – краткая и четкая формулировка того, чего должен достичь данный тест;

– конфигурация средств тестирования – спецификация ввода, спецификация вывода, условия испытаний;

– настройка на прогон теста – эта процедура подобна методике тестирования. Она предусматривает описание действий, выполняемых тестировщиком, и ожидаемых результатов. Если настройки автоматизированы, это может выглядеть как **run setupSC03.pl;**

– методика тестирования – описание действий, выполняемых тестировщиком, и ожидаемых результатов;

– взаимозависимость тестовых случаев – идентификация любого тестового случая, прогон которого должен предшествовать прогону данного теста, дабы выполнение данного теста начиналось при заданных условиях;

– очистка теста – если системы была переведена в неустойчивое состояние или данные оказались разрушенными, очистка предоставит шанс устранить подобные ситуации.

Управление конфигурацией тестового случая

Во время прогона теста необходимо выполнить заданный набор операций на заданной конфигурации системы. Это обстоятельство гарантирует, что тест можно воспроизвести и что он осуществляет проверку заданных функциональных возможностей. По мере того, как меняется программный продукт в результате исправления обнаруженных дефектов или по причине изменений, внесенных в технические требования, тесты также требуют изменений. Функциональные возможности программного продукта и тесты должны синхронно реагировать на такие изменения, в противном случае тесты приведут к неверным результатам.

Пересмотр и отладка тестов

После написания или автоматизации тест необходимо проверить на наличие дефектов с целью их немедленного устранения, и затем испытать на некоторой сборке программного продукта. Статическое тестирование тестовых случаев можно проводить с применением той же технологии, которая используется для проверки выполнения технических требований, т.е. обследования, сквозного контроля или экспертных оценок. Эти методы могут применяться в различных сочетаниях, если сложность тестов такова, что

необходимо тщательное статическое тестирование. Например, экспертные оценки можно использовать в методиках, а формальное обследование – в автоматизированных сценариях, реализующих методики тестирования.

При проверке тестовых случаев необходимо обратить внимание на ряд следующих моментов:

– В какой мере соответствует тест или тестовый набор функциональным возможностям, заявленным в технических требованиях?

– Покрывает ли тестовый набор все технические требования?

– Организованы ли тесты достаточно эффективно, чтобы можно было обходиться минимальными конфигурациями средств тестирования?

– Включены ли тесты в систему управления конфигурациями?

– Есть ли среди тестов избыточные? Можно ли устранить эту избыточность?

– Достаточно ли подробно разработана методика тестирования, чтобы стала возможной ее автоматизация?

– Снабжен ли каждый шаг тестирования четко определенными ожидаемыми результатами (критерий удачного/неудачного исхода испытаний)?

– Правильно ли воспроизводят автоматизированные тесты неавтоматизированные шаги тестирования?

Если вы провели статическое тестирование плана проведения испытаний, то пересмотр тестовых случаев пойдет намного проще, поскольку вопросы, касающиеся покрытий технических требований тестами, должны решаться во время пересмотра плана проведения испытаний. Вопросы, касающиеся избыточности тестов и эффективного использования конфигурации средств тестирования, также должны быть затронуты в процессе пересмотра плана проведения испытаний, во всяком случае, на уровне предварительных решений. Тщательные пересмотры планов проведения испытаний и тестовых случаев, равно как и статическое тестирование программного продукта, содействуют раннему обнаружению дефектов в тестах и позволяют сократить затраты времени на отладку тестовых случаев.

В рамках такого пересмотра каждый новый тест нужно прогнать на некоторой сборке программного продукта, дабы удостовериться, что методика тестирования позволяет получить ожидаемые результаты. Поскольку этот тест, по-видимому, будет прогоняться на программном коде, в изобилии содержащем дефекты, необходимо соблюдать определенную осторожность при анализе неудачных исходов прогона теста в плане, что является источником проблемы – программный код или сам тест. Отладка тестового случая часто требует от исполнителей такого же инженерного искусства и аналитических способностей, которые так необходимы разработчикам программного обеспечения во время отладки кода программного продукта.

Практическая часть

Содержание задания

Спроектировать и разработать тесты для программного продукта из предыдущей лабораторной работы на основании выдвинутых требований. Проектов тестов должно быть не меньше, чем требований в лабораторной работе №9, а тестовых случаев не менее трех на каждый спроектированный тест. Составить соответствующую документацию. Пример составления документа проектирования и разработки тестов приведен в книге «Быстрое тестирование», страницы 328 – 367 (книга находится в папке с заданием на лабораторную работу).

Порядок выполнения работы

1 Ознакомиться с теоретической частью.

2 Спроектировать и разработать тесты.

3 Составить соответствующую документацию (**обязательными являются ВСЕ пункты указанные ниже в содержании документа**).

4 Показать работу преподавателю.

5 После защиты, выполненного задания, скинуть отчёт в Google Classroom (в **названии** указать **фамилию и номер лабораторной**, например, Ivanov10).

Содержание документа

1 Титульный лист (**на титульном листе вместо варианта указать название тестируемого программного продукта**).

2 Введение.

3 Свойства, которые должны тестироваться.

4 Проекты тестов.

5 Тестовые случаи.

6 Вывод о проделанной работе.

Защита работ проводится индивидуально.

Контрольные вопросы

1 Входные данные для проектирования и разработки тестов.

2 Выходные данные для проектирования и разработки тестов.

3 Что такое модульность (с точки зрения тестирования)?

4 Из каких компонентов состоит тест?

5 В чем смысл такой технологии проектирования тестов, как «Разбиение на классы эквивалентности»?

6 Перечислите основные элементы шаблона тестового случая.