

Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

Н. Б. ОСИПЕНКО, А. Н. ОСИПЕНКО

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Практическое руководство

для студентов специальности I–40 01 01
«Программное обеспечение информационных технологий»

Гомель
ГГУ им. Ф. Скорины
2015

УДК 004.415.53-057.4
ББК 32.972я73
О-74

Рецензенты:

кандидат физико-математических наук А. И. Рябченко;
кандидат физико-математических наук Д. С. Кузьменков

Рекомендовано к изданию научно-методическим советом
учреждения образования «Гомельский государственный
университет имени Франциска Скорины»

Осипенко, Н. Б.

О-74 Тестирование программного обеспечения : практическое
руководство / Н. Б. Осипенко, А. Н. Осипенко ; М-во образования
РБ, Гом. гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им.
Ф. Скорины, 2015. – 47 с.
ISBN 978-985-577-057-3

Практическое руководство предназначено для оказания помощи
студентам в усвоении знаний по разделу «Тестирование программного
обеспечения» курса «Основы стандартизации и сертификации программ-
ного обеспечения». В него включены также контрольные вопросы по
данному разделу.

Адресовано студентам специальности 1–40 01 01 «Программное
обеспечение информационных технологий».

УДК 004.415.53-057.4
ББК 32.972я73

ISBN 978-985-577-057-3

© Осипенко Н. Б., Осипенко А. Н., 2015
© Учреждение образования «Гомельский
государственный университет
имени Франциска Скорины», 2015

Оглавление

Предисловие.....	3
1 Тестирование программного обеспечения.....	5
1.1 Основные понятия тестирования программного обеспечения.....	5
1.1.1 Проблематика тестирования программного обеспечения	5
1.1.2 Основные определения.....	6
1.1.3 Экономика тестирования.....	7
1.1.4 Аксиомы (принципы) тестирования	9
Вопросы и задания для самоконтроля.....	10
1.2 Тестирование надежности программного обеспечения.....	10
1.2.1 Философия тестирования	10
1.2.2 Тестирование модулей.....	12
1.2.3 Комплексное тестирование	19
1.2.4 Организация и этапы тестирования при испытаниях надежности сложных программных средств.....	20
Вопросы и задания для самоконтроля.....	27
1.3 Тестирование программного обеспечения.....	28
1.3.1 Тестирование программного обеспечения.....	28
1.3.2 Место и цель этапа тестирования программного обеспечения	29
1.3.3 Виды тестирования	32
1.3.4 Передовые технологии в тестировании (автоматизация тестирования)	33
Вопросы и задания для самоконтроля.....	34
1.4 Виды тестирования программного обеспечения	35
1.4.1 Функциональные виды тестирования.....	35
1.4.2 Нефункциональные виды тестирования.....	37
1.4.3 Связанные с изменениями виды тестирования.....	38
1.4.4 Тестирование удобства пользования	41
1.4.5 Тестирование на отказ и восстановление.....	43
1.4.6 Конфигурационное тестирование	43
Вопросы и задания для самоконтроля.....	45
Литература	47

Предисловие

Целью данного практического руководства является оказание помощи студентам в усвоении знаний и формировании практических навыков при изучении раздела «Тестирование программного обеспечения» курса «Основы стандартизации и сертификации программного обеспечения».

Тестирование программного обеспечения – это процесс исследования программного продукта с двумя целями: во-первых, продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям и, во-вторых, выявить ситуации, в которых поведение программы неправильно, нежелательно или не соответствует спецификации.

Существующие на сегодня методы тестирования программного обеспечения не позволяют однозначно и полностью выявить все дефекты и установить корректность функционирования анализируемой программы, поэтому все существующие методы тестирования действуют в рамках формального процесса проверки исследуемого или разрабатываемого программного обеспечения. Процесс формальной проверки (верификация) может доказать, что дефекты отсутствуют с точки зрения используемого метода.

Существует множество подходов к решению задачи тестирования и верификации программного обеспечения, но эффективное тестирование сложных программных продуктов – это процесс в высшей степени творческий, не сводящийся к следованию строгим и чётким процедурам или созданию таковых.

Состав и содержание документации, сопутствующей процессу тестирования, определяется стандартом IEEE 829-1998.

В практическое руководство включены теоретические сведения по четырем темам: «Основные понятия тестирования программного обеспечения», «Тестирование надежности программного обеспечения», «Тестирование программного обеспечения» и «Виды тестирования программного обеспечения», а также вопросы для самоконтроля, усвоение которых необходимо для выполнения заданий на практических занятиях.

Практическое руководство адресовано студентам специальности 1–40 01 01 «Программное обеспечение информационных технологий».

1 Тестирование программного обеспечения

1.1 Основные понятия тестирования программного обеспечения

1.1.1 Проблематика тестирования программного обеспечения

Многие организации, занимающиеся созданием программного обеспечения, до 50 % средств, выделенных на разработку программ, тратят на тестирование, что составляет миллиарды долларов по всему миру в целом. И все же, несмотря на громадные капиталовложения, знаний о сути тестирования явно не хватает, и большинство программных продуктов неприемлемо, ненадежно даже после «основательного тестирования».

О состоянии дел лучше всего свидетельствует тот факт, что большинство людей, работающих в области обработки данных, даже не могут правильно определить понятие «тестирование», и это на самом деле главная причина неудач. Если попросить любого профессионала определить понятие «тестирование» либо открыть (как правило, слишком краткую) главу о тестировании любого учебника программирования, то скорее всего можно встретить такое определение: «Тестирование – процесс, подтверждающий правильность программы и демонстрирующий, что ошибок в программе нет». Основной недостаток подобного определения заключается в том, что оно совершенно неправильно; фактически это почти определение антонима слова «тестирование». Поэтому определение описывает невыполнимую задачу, а так как тестирование зачастую все же выполняется с успехом, по крайней мере, с некоторым успехом, то такое определение логически некорректно.

Правильное определение тестирования таково: ***Тестирование – процесс выполнения программы с намерением найти ошибки.***

Тестирование оказывается довольно необычным процессом (вот почему оно и считается трудным), так как это процесс разрушительный. Ведь цель проверяющего (тестовика) – заставить программу сбиться. Он доволен, если это ему удастся; если же программа на его тесте не сбивается, он не удовлетворен.

Невозможно гарантировать отсутствие ошибок в программе; в лучшем случае можно попытаться показать наличие ошибок. Если

программа правильно ведет себя для значительного набора тестов, нет оснований утверждать, что в ней нет ошибок; со всей определенностью можно лишь утверждать, что неизвестно, когда эта программа не работает. Конечно, если есть причины считать данный набор тестов способным с большой вероятностью обнаружить все возможные ошибки, то можно говорить о некотором уровне уверенности в правильности программы, устанавливаемой этими тестами.

О тестировании говорить довольно трудно, поскольку, хотя оно и способствует повышению надежности программного обеспечения, его значение ограничено. Надежность невозможно внести в программу в результате тестирования, она определяется правильностью этапов проектирования. Наилучшее решение проблемы надежности – с самого начала не допускать ошибок в программе. Однако вероятность того, что удастся безупречно спроектировать большую программу, бесконечно мала. Роль тестирования состоит как раз в том, чтобы определить местонахождение немногочисленных ошибок, оставшихся в хорошо спроектированной программе. Попытки с помощью тестирования достичь надежности плохо спроектированной программы совершенно бесплодны.

1.1.2 Основные определения

В тестировании можно выделить несколько различных процессов, некоторые из которых «тестирование», «отладка», «доказательство», «контроль», «испытание» и др. зачастую некорректно используются как синонимы. Приведем определения форм тестирования, различающихся по отношению к *среде*, на которую оно опирается.

Тестирование (testing) – процесс выполнения программы (или части программы) с намерением (или целью) найти ошибки.

Доказательство (proof) – попытка найти ошибки в программе безотносительно к внешней для программы среде. Большинство методов доказательства предполагает формулировку утверждений о поведении программы и затем вывод и доказательство математических теорем о правильности программы. Доказательства могут рассматриваться как форма тестирования, хотя они и не предполагают прямого выполнения программы. Многие исследователи считают доказательство альтернативой тестированию – взгляд во многом ошибочный.

Контроль (verification) – попытка найти ошибки, выполняя программу в тестовой, или моделируемой, среде.

Испытание (validation) – попытка найти ошибки, выполняя программу в заданной реальной среде.

Аттестация (certification) – авторитетное подтверждение правильности программы. При тестировании с целью аттестации выполняется сравнение с некоторым заранее определенным стандартом.

Отладка (debugging) не является разновидностью тестирования. Хотя слова «отладка» и «тестирование» часто используются как синонимы, под ними подразумеваются разные виды деятельности. Тестирование – деятельность, направленная на обнаружение ошибок; отладка направлена на установление точной природы известной ошибки, а затем – на исправление этой ошибки. Эти два вида деятельности связаны – результаты тестирования являются исходными данными для отладки.

Другой ряд определений, приведенный ниже, охватывает вторую сторону тестирования: *типы ошибок*, которые предполагается обнаружить, и стандарты, с которыми сопоставляются тестируемые программы.

Тестирование *модуля*, или автономное тестирование (module testing, unit testing), – контроль отдельного программного модуля, обычно в изолированной среде (т. е. изолированно от всех остальных модулей). Тестирование модуля иногда включает также математическое доказательство.

Тестирование *сопряжений* (integration testing) – контроль сопряжений между частями системы (модулями, компонентами, подсистемами).

Тестирование *внешних функций* (external function testing) – контроль внешнего поведения системы, определенного внешними спецификациями.

Комплексное тестирование (system testing) – контроль и/или испытание системы по отношению к исходным целям. Комплексное тестирование является процессом контроля, если оно выполняется в моделируемой среде, и процессом испытания, если выполняется в среде реальной, жизненной.

Тестирование *приемлемости* (acceptance testing) – проверка соответствия программы требованиям пользователя.

Тестирование *настройки* (installation testing) – проверка соответствия каждого конкретного варианта установки системы с целью выявить любые ошибки, возникшие в процессе настройки системы.

1.1.3 Экономика тестирования

В общем случае невозможно обнаружить все ошибки программы. А это в свою очередь порождает экономические проблемы, задачи,

связанные с функциями человека в процессе отладки, способы построения тестов.

Одним из способов изучения поставленного вопроса является исследование стратегии тестирования, называемой *стратегией «черного ящика»*, тестированием с управлением по данным или тестированием с управлением по входу–выходу. При использовании этой стратегии программа рассматривается как «черный ящик». Иными словами, такое тестирование имеет целью выяснение обстоятельств, в которых поведение программы не соответствует спецификации. Тестовые же данные используются только в соответствии со спецификацией программы (т. е. без учета знаний о ее внутренней структуре).

При таком подходе обнаружение всех ошибок в программе является критерием исчерпывающего входного тестирования. Последнее может быть достигнуто, если в качестве тестовых наборов использовать все возможные наборы входных данных. Если такое испытание представляется сложным, то еще сложнее создать исчерпывающий тест для большой программы. Образно говоря, число тестов можно оценить «числом большим, чем бесконечность».

Построение исчерпывающего входного теста невозможно. Это подтверждается двумя аргументами: во-первых, нельзя создать тест, гарантирующий отсутствие ошибок; во-вторых, разработка таких тестов противоречит экономическим требованиям. Поскольку исчерпывающее тестирование исключается, нашей целью должна стать максимизация результативности капиталовложений в тестирование (иными словами, максимизация числа ошибок, обнаруживаемых одним тестом). Для этого мы можем рассматривать внутреннюю структуру программы и делать некоторые разумные, но, конечно, не обладающие полной гарантией достоверности предположения.

Стратегия «белого ящика», или стратегия тестирования, управляемого логикой программы, позволяет исследовать внутреннюю структуру программы. В этом случае тестирующий получает тестовые данные путем анализа логики программы (к сожалению, здесь часто не используется спецификация программы).

Сравним способ построения тестов при данной стратегии с исчерпывающим входным тестированием стратегии «черного ящика». Непосвященному может показаться, что достаточно построить такой набор тестов, в котором каждый оператор выполняется хотя бы один раз; нетрудно показать, что это неверно. Не вдаваясь в детали, укажем лишь, что исчерпывающему входному тестированию может быть поставлено в соответствие исчерпывающее тестирование маршрутов. Подразумевается, что программа проверена полностью, если с помощью тестов

удается осуществить выполнение программы по всем возможным маршрутам ее потока (графа) передач управления.

Последнее утверждение имеет два слабых пункта. Первый из них состоит в том, что число не повторяющих друг друга маршрутов в программе – астрономическое. Второй слабый пункт утверждения заключается в том, что, хотя исчерпывающее тестирование маршрутов является полным тестом и хотя каждый маршрут программы может быть проверен, сама программа будет содержать ошибки. Это объясняется следующим образом. Во-первых, исчерпывающее тестирование маршрутов не может дать гарантии того, что программа соответствует описанию. Например, вместо требуемой программы сортировки по возрастанию случайно была написана программа сортировки по убыванию. В этом случае ценность тестирования маршрутов невелика, поскольку после тестирования в программе окажется одна ошибка, т. е. программа неверна.

Во-вторых, программа может быть неверной в силу того, что пропущены некоторые маршруты. Исчерпывающее тестирование маршрутов не обнаружит их отсутствия.

В-третьих, исчерпывающее тестирование маршрутов не может обнаружить ошибок, появление которых зависит от обрабатываемых данных.

1.1.4 Аксиомы (принципы) тестирования

Хорош тот тест, для которого высока вероятность обнаружить ошибку. Одна из самых сложных проблем при тестировании – решить, когда нужно его закончить. Необходимая часть всякого теста – описание ожидаемых выходных данных или результатов. Избегайте невоспроизводимых тестов, не тестируйте «с лету». Готовьте тесты как для правильных, так и для неправильных входных данных. Детально изучите результаты каждого теста. По мере того как число ошибок, обнаруженных в некотором компоненте программного обеспечения, увеличивается, растет относительная вероятность существования в нем необнаруженных ошибок. Поручайте тестирование самым способным программистам. Считайте тестируемость ключевой задачей вашей разработки. Проект системы должен быть таким, чтобы каждый модуль подключался к системе только один раз. Никогда не изменяйте программу, чтобы облегчить ее тестирование. Тестирование, как почти всякая другая деятельность, должно начинаться с постановки целей.

Наиболее важными принципами тестирования являются следующие:

1) тестирование – это процесс выполнения программ с целью обнаружения ошибок; 2) хорошим считается тест, который имеет высокую вероятность обнаружения еще не выявленной ошибки; 3) удачным считается тест, который обнаруживает еще не выявленную ошибку.

Вопросы и задания для самоконтроля

- 1 Подтверждает ли тестирование правильность программы?
- 2 Что можно сказать о программе, если она на значительном количестве тестов ведет себя правильно?
- 3 Может ли повысить надежность программы процесс тестирования?
- 4 Приведите примеры понятий, некорректно используемых как синонимы процесса «тестирования».
- 5 Дайте классификацию различных форм тестирования по отношению к среде, на которую оно опирается.
- 6 В чём отличия процессов тестирования от отладки?
- 7 В чём отличия процессов контроля от испытания?
- 8 Перечислите типы ошибок, обнаруживаемые при тестировании.
- 9 Как иногда называют тестирование с управлением по данным?
- 10 Используется ли информация о внутренней структуре программы при тестировании на основании стратегии «черного ящика»?
- 11 Можно ли построить исчерпывающий входной тест? Почему?
- 12 Что, как правило, не учитывает стратегия «белого ящика» при тестировании программы?
- 13 Перечислите стратегии тестирования.
- 14 Каковы плюсы и минусы исчерпывающего тестирования маршрутов?
- 15 Каковы основные принципы тестирования?

1.2 Тестирование надежности программного обеспечения

1.2.1 Философия тестирования

Тестирование программного обеспечения охватывает ряд видов деятельности, аналогичный последовательности процессов разработки

программного обеспечения. Сюда входят постановка задачи для теста, проектирование, написание тестов, тестирование тестов, выполнение тестов и изучение результатов тестирования. Решающую роль играет проектирование теста. Возможен целый спектр подходов к выработке философии, или стратегии проектирования. Чтобы ориентироваться в стратегиях проектирования тестов, стоит рассмотреть два крайних подхода, находящихся на границах спектра. Следует отметить также, что многие из тех, кто работает в этой области, часто бросаются в одну или другую крайность.

Сторонник подхода, соответствующего левой границе спектра (рисунок 1.1), проектирует свои тесты, исследуя внешние спецификации или спецификации сопряжения программы или модуля, которые он тестирует. Программу он рассматривает как «черный ящик». Позиция его такова: «Меня не интересует, как выглядит эта программа и выполнил ли я все команды или все пути. Я буду удовлетворен, если программа будет вести себя так, как указано в спецификациях». Его идеал – проверить все возможные комбинации и значения на входе.

Приверженец подхода, соответствующего другому концу спектра, проектирует свои тесты, изучая логику программы. Он начинает с того, что стремится подготовить достаточное число тестов для того, чтобы каждая команда была выполнена, по крайней мере, один раз. Если он немного более искушен, то проектирует тесты так, чтобы каждая команда условного перехода выполнялась в каждом направлении, хотя бы раз. Его идеал – проверить каждый путь, каждую ветвь алгоритма. При этом его совсем (или почти совсем) не интересуют спецификации.



Рисунок 1.1 – Схема спектра подходов к проектированию тестов

Ни одна из этих крайностей не является хорошей стратегией. Первая из них, а именно та, в соответствии с которой программа рассматривается как «черный ящик», предпочтительней. К сожалению, она страдает тем недостатком, что совершенно неосуществима. Рассмотрим попытку тестирования тривиальной программы, получающей на входе три числа и вычисляющей их среднее арифметическое. Тестирование этой программы для всех значений входных данных невозможно. Даже

для машины с относительно низкой точностью вычислений количество тестов исчислялось бы миллиардами. Имея вычислительную мощность, достаточную для выполнения всех тестов в разумное время, мы потратили бы на несколько порядков больше времени для того, чтобы эти тесты подготовить, а затем проверить. Такие программы, как системы реального времени, операционные системы и программы управления данными, которые сохраняют «память» о предыдущих входных данных, еще хуже. Нам потребовалось бы тестировать программу не только для каждого входного значения, но и для каждой последовательности, каждой комбинации входных данных. Поэтому исчерпывающее тестирование для всех входных данных любой программы неосуществимо.

Эти рассуждения приводят ко второму фундаментальному принципу тестирования: тестирование – проблема в значительной степени экономическая. Поскольку исчерпывающее тестирование невозможно, мы должны ограничиться чем-то меньшим. Каждый тест должен давать максимальную отдачу по сравнению с затратами. Эта отдача измеряется вероятностью того, что тест выявит не обнаруженную прежде ошибку. Затраты измеряются временем и стоимостью подготовки, выполнения и проверки результатов теста. Считая, что затраты ограничены бюджетом и графиком, можно утверждать, что искусство тестирования, по существу, представляет собой искусство отбора тестов с максимальной отдачей. Каждый тест должен быть представителем некоторого класса входных значений, так чтобы его правильное выполнение создавало у нас некоторую убежденность в том, что для определенного класса входных данных программа будет выполняться правильно. Это обычно требует некоторого знания алгоритма и структуры программы, и мы, таким образом, смещаемся к правому концу спектра.

1.2.2 Тестирование модулей

Вторым по важности аспектом тестирования после проектирования тестов является *последовательность слияния всех модулей* в систему или программу. Эта сторона вопроса обычно не получает достаточного внимания и часто рассматривается слишком поздно. Выбор этой последовательности, однако, является одним из самых жизненно важных решений, принимаемых на этапе тестирования, поскольку он определяет форму, в которой записываются тесты, типы необходимых инструментов тестирования, последовательность программирования

модулей, а также тщательность и экономичность всего этапа тестирования. По этой причине такое решение должно приниматься на уровне проекта в целом и на достаточно ранней его стадии.

Тестирование модулей (или блоков) представляет собой процесс тестирования отдельных подпрограмм или процедур программы. Здесь подразумевается, что, прежде чем начинать тестирование программы в целом, следует протестировать отдельные небольшие модули, образующие эту программу. Такой подход мотивируется тремя причинами. Во-первых, появляется возможность управлять комбинаторикой тестирования, поскольку первоначально внимание концентрируется на небольших модулях программы. Во-вторых, облегчается задача отладки программы, т. е. обнаружение места ошибки и исправление текста программы. В-третьих, допускается параллелизм, что позволяет одновременно тестировать несколько модулей.

Цель тестирования модулей – сравнение функций, реализуемых модулем, со спецификациями его функций или интерфейса.

Тестирование модулей в основном ориентировано на принцип «белого ящика». Это объясняется прежде всего тем, что принцип «белого ящика» труднее реализовать при переходе в последующем к тестированию более крупных единиц, например программ в целом. Кроме того, последующие этапы тестирования ориентированы на обнаружение ошибок различного типа, т. е. ошибок, не обязательно связанных с логикой программы, а возникающих, например, из-за несоответствия программы требованиям пользователя.

Имеется большой выбор возможных подходов, которые могут быть использованы для слияния модулей в более крупные единицы. В большинстве своем они могут рассматриваться как варианты шести основных подходов: пошаговое тестирование; восходящее тестирование; нисходящее тестирование; метод «большого скачка»; метод сэндвича; модифицированный метод сэндвича.

Метод сэндвича. Тестирование методом сэндвича представляет собой компромисс между восходящим и нисходящим подходами. Здесь делается попытка воспользоваться достоинствами обоих методов, избежав их недостатков. При использовании этого метода одновременно начинают восходящее и нисходящее тестирование, собирая программу снизу и сверху, встречаясь где-то в середине. Точка встречи зависит от конкретной тестируемой программы и должна быть заранее определена при изучении ее структуры. Например, если разработчик может представить свою систему в виде уровня прикладных модулей, затем уровня модулей обработки запросов, затем уровня примитивных функций, то он может решить применять нисходящий метод на уровне

прикладных модулей (программируя заглушки вместо модулей обработки запросов), а на остальных уровнях применить восходящий метод. Метод сэндвича сохраняет такое достоинство нисходящего и восходящего подходов, как начало интеграции системы на самом раннем этапе. Поскольку вершина программы вступает в строй рано, мы, как в нисходящем методе, уже на раннем этапе получаем работающий каркас программы. Так как нижние уровни программы создаются восходящим методом, снимаются проблемы нисходящего метода, которые были связаны с невозможностью тестировать некоторые условия в глубине программы.

Модифицированный метод сэндвича. При тестировании методом сэндвича возникает та же проблема, что и при нисходящем подходе, но не так остро. Проблема эта в том, что невозможно досконально тестировать отдельные модули. Восходящий этап тестирования по методу сэндвича решает эту проблему для модулей нижних уровней, но она может по-прежнему оставаться открытой для нижней половины верхней части программы. В модифицированном методе сэндвича нижние уровни также тестируются строго снизу вверх. А модули верхних уровней сначала тестируются изолированно, а затем собираются нисходящим методом. Таким образом, модифицированный метод сэндвича также представляет собой компромисс между восходящим и нисходящим подходами.

Восходящее тестирование. Программа собирается и тестируется «снизу вверх». Только модули самого нижнего уровня («терминальные» модули; модули, не вызывающие других модулей) тестируются изолированно, автономно. После того как тестирование этих модулей завершено, вызов их должен быть так же надежен, как вызов встроенной функции языка или оператор присваивания. Затем тестируются модули, непосредственно вызывающие уже проверенные. Эти модули более высокого уровня тестируются не автономно, а вместе с уже проверенными модулями более низкого уровня. Процесс повторяется до тех пор, пока не будет достигнута вершина. Здесь завершается и тестирование модулей, и тестирование сопряжений программы.

При восходящем тестировании для каждого модуля необходим драйвер: нужно подавать тесты в соответствии с сопряжением тестируемого модуля. Одно из возможных решений – написать для каждого модуля небольшую ведущую программу. Тестовые данные представляются как «встроенные» непосредственно в эту программу переменные и структуры данных, и она многократно вызывает тестируемый модуль, с каждым вызовом передавая ему новые тестовые данные. Имеется и лучшее решение: воспользоваться программой тестирования

модулей – это инструмент тестирования, позволяющий описывать тесты на специальном языке и избавляющий от необходимости писать драйверы.

Здесь отсутствуют проблемы, связанные с невозможностью или трудностью создания всех тестовых ситуаций, характерные для нисходящего тестирования. Драйвер как средство тестирования применяется непосредственно к тому модулю, который тестируется, где нет промежуточных модулей, которые следует принимать во внимание. Анализируя другие проблемы, возникающие при нисходящем тестировании, можно заметить, что при восходящем тестировании невозможно принять неразумное решение о совмещении тестирования с проектированием программы, поскольку нельзя начать тестирование до тех пор, пока не спроектированы модули нижнего уровня. Не существует также и трудностей с незавершенностью тестирования одного модуля при переходе к тестированию другого, потому что при восходящем тестировании с применением нескольких версий заглушки нет сложностей с представлением тестовых данных.

Нисходящее тестирование. Нисходящее тестирование (называемое также нисходящей разработкой) не является полной противоположностью восходящему, но в первом приближении может рассматриваться как таковое. При нисходящем подходе программа собирается и тестируется «сверху вниз». Изолированно тестируется только головной модуль. После того как тестирование этого модуля завершено, с ним соединяются (например, редактором связей) один за другим модули, непосредственно вызываемые им, и тестируется полученная комбинация. Процесс повторяется до тех пор, пока не будут собраны и проверены все модули.

При этом подходе возникают два вопроса: 1. «Что делать, когда тестируемый модуль вызывает модуль более низкого уровня (которого в данный момент еще не существует)?» и 2. «Как подаются тестовые данные?»

Ответ на первый вопрос состоит в том, что для имитации функций недостающих модулей программируются модули – «заглушки», которые моделируют функции отсутствующих модулей.

Интересен и второй вопрос: «В какой форме готовятся тестовые данные и как они передаются программе?» Если бы головной модуль содержал все нужные операции ввода и вывода, ответ был бы прост: тесты пишутся в виде обычных для пользователей внешних данных и передаются программе через выделенные ей устройства ввода. Так, однако, случается редко. В хорошо спроектированной программе физические операции ввода-вывода выполняются на нижних уровнях

структуры, поскольку физический ввод– вывод – абстракция довольно низкого уровня. Поэтому для того, чтобы решить проблему экономически эффективно, модули добавляются не в строго нисходящей последовательности (все модули одного горизонтального уровня, затем модули следующего уровня), а таким образом, чтобы обеспечить функционирование операций физического ввода–вывода как можно быстрее. Когда эта цель достигнута, нисходящее тестирование получает значительное преимущество: все дальнейшие тесты готовятся в той же форме, которая рассчитана на пользователя.

Нисходящий метод имеет как достоинства, так и недостатки по сравнению с восходящим. Самое значительное достоинство – то, что этот метод совмещает тестирование модуля, тестирование сопряжений и частично тестирование внешних функций. С этим же связано другое его достоинство: когда модули ввода–вывода уже подключены, тесты можно готовить в удобном виде. Нисходящий подход выгоден также в том случае, когда есть сомнения относительно осуществимости программы в целом или когда в проекте программы могут оказаться серьезные дефекты.

Преимуществом нисходящего подхода очень часто считают отсутствие необходимости в драйверах; вместо драйверов вам просто следует написать «заглушки».

Нисходящий метод тестирования имеет, к сожалению, некоторые недостатки. Основным из них является то, что модуль редко тестируется досконально сразу после его подключения. Дело в том, что основательное тестирование некоторых модулей может потребовать крайне изощренных заглушек. Программист часто решает не тратить массу времени на их программирование, а вместо этого пишет простые заглушки и проверяет лишь часть условий в модуле. Он, конечно, собирается вернуться и закончить тестирование рассматриваемого модуля позже, когда уберет заглушки. Такой план тестирования – определенно не лучшее решение, поскольку об отложенных условиях часто забывают.

Второй тонкий недостаток нисходящего подхода состоит в том, что он может породить веру в возможность начать программирование и тестирование верхнего уровня программы до того, как вся программа будет полностью спроектирована. Эта идея на первый взгляд кажется экономичной, но обычно дело обстоит совсем наоборот. Большинство опытных проектировщиков признает, что проектирование программы – процесс итеративный. Редко первый проект оказывается совершенным. Нормальный стиль проектирования структуры программы предполагает по окончании проектирования нижних уровней вернуться

назад и подправить верхний уровень, внося в него некоторые усовершенствования или исправляя ошибки, либо иногда даже выбросить проект и начать все сначала, потому что разработчик внезапно увидел лучший подход. Если же головная часть программы уже запрограммирована и оттестирована, то возникает серьезное сопротивление любым улучшениям ее структуры. В конечном итоге за счет таких улучшений обычно можно сэкономить больше, чем те несколько дней или недель, которые рассчитывает выиграть проектировщик, приступая к программированию слишком рано.

Метод «большого скачка». Вероятно, самый распространенный подход к интеграции модулей – метод «большого скачка». В соответствии с этим методом каждый модуль тестируется автономно. По окончании тестирования модулей они интегрируются в систему все сразу. Метод «большого скачка» по сравнению с другими подходами имеет много недостатков и мало достоинств. Заглушки и драйверы необходимы для каждого модуля. Модули не интегрируются до самого последнего момента, а это означает, что в течение долгого времени серьезные ошибки в сопряжениях могут остаться необнаруженными. Если программа мала (как, например, программа загрузчика) и хорошо спроектирована, метод «большого скачка» может оказаться приемлемым. Однако для крупных программ метод «большого скачка» обычно губителен.

Пошаговое тестирование. Реализация процесса тестирования модулей опирается на два ключевых положения: построение эффективного набора тестов и выбор способа, посредством которого модули комбинируются при построении из них рабочей программы. Второе положение является важным, так как оно задает форму написания тестов модуля, типы средств, используемых при тестировании, порядок кодирования и тестирования модулей, стоимость генерации тестов и стоимость отладки. Рассмотрим два подхода к комбинированию модулей: пошаговое и монолитное тестирование.

Возникает вопрос: «Что лучше – выполнить по отдельности тестирование каждого модуля, а затем, комбинируя их, сформировать рабочую программу или же каждый модуль для тестирования подключать к набору ранее оттестированных модулей?». Первый подход обычно называют монолитным методом, или методом «большого удара», при тестировании и сборке программы; второй подход известен как пошаговый метод тестирования или сборки.

Метод пошагового тестирования предполагает, что модули тестируются не изолированно друг от друга, а подключаются поочередно для выполнения теста к набору уже ранее оттестированных модулей.

Пошаговый процесс продолжается до тех пор, пока к набору оттестированных модулей не будет подключен последний модуль.

Детального разбора обоих методов мы делать не будем, приведем лишь некоторые общие выводы.

1. Монолитное тестирование требует больших затрат труда. При пошаговом же тестировании «снизу–вверх» затраты труда сокращаются.

2. Расход машинного времени при монолитном тестировании меньше.

3. Использование монолитного метода предоставляет большие возможности для параллельной организации работы на начальной фазе тестирования (тестирования всех модулей одновременно). Это положение может иметь важное значение при выполнении больших проектов, в которых много модулей и много исполнителей, поскольку численность персонала, участвующего в проекте, максимальна на начальной фазе.

4. При пошаговом тестировании раньше обнаруживаются ошибки в интерфейсах между модулями, поскольку раньше начинается сборка программы. В противоположность этому при монолитном тестировании модули «не видят друг друга» до последней фазы процесса тестирования.

5. Отладка программ при пошаговом тестировании легче. Если есть ошибки в межмодульных интерфейсах, а обычно так и бывает, то при монолитном тестировании они могут быть обнаружены лишь тогда, когда собрана вся программа. В этот момент локализовать ошибку довольно трудно, поскольку она может находиться в любом месте программы. Напротив, при пошаговом тестировании ошибки такого типа в основном связаны с тем модулем, который подключается последним.

6. Результаты пошагового тестирования более совершенны.

В заключение отметим, что п. 1, 4, 5, 6 демонстрируют преимущества пошагового тестирования, а п. 2 и 3 – его недостатки. Поскольку для современного этапа развития вычислительной техники характерны тенденции к уменьшению стоимости аппаратуры и увеличению стоимости труда, последствия ошибок в математическом обеспечении весьма серьезны, а стоимость устранения ошибки тем меньше, чем раньше она обнаружена; преимущества, указанные в п. 1, 4, 5, 6, выступают на первый план. В то же время ущерб, наносимый недостатками (п. 2 и 3), невелик. Все это позволяет нам сделать вывод, что пошаговое тестирование является предпочтительным.

Убедившись в преимуществах пошагового тестирования перед монолитным, исследуем две возможные стратегии тестирования: нисходящее и восходящее. Прежде всего внесем ясность в терминологию.

Во-первых, термины «нисходящее тестирование», «нисходящая разработка», «нисходящее проектирование» часто используются как синонимы. Действительно, термины «нисходящее тестирование» и «нисходящая разработка» являются синонимами (в том смысле, что они подразумевают определенную стратегию при тестировании и создании текстов модулей), но нисходящее проектирование – это совершенно иной и независимый процесс. Программа, спроектированная нисходящим методом, может тестироваться и нисходящим, и восходящим методами.

Во-вторых, восходящая разработка, или тестирование, часто отождествляется с монокитным тестированием. Это недоразумение возникает из-за того, что начало восходящего тестирования идентично монокитному при тестировании нижних или терминальных модулей. Но выше было показано, что восходящее тестирование на самом деле представляет собой пошаговую стратегию.

1.2.3 Комплексное тестирование

Комплексное тестирование не является тестированием всех функций полностью собранной системы; тестирование такого типа называется тестированием внешних функций. Комплексное тестирование – процесс поисков несоответствия системы ее исходным целям. Элементами, участвующими в комплексном тестировании, служат сама система, описание целей продукта и вся документация, которая будет поставляться вместе с системой. Внешние спецификации, которые были ключевым элементом тестирования внешних функций, играют лишь незначительную роль в комплексном тестировании.

Если вы не сформулировали цели вашего продукта или если эти цели неизмеримы, вы не можете выполнить комплексное тестирование.

Комплексное тестирование может быть процессом и контроля, и испытаний. Процессом испытаний оно является тогда, когда выполняется в реальной среде пользователя или в обстановке, которая специально создана так, чтобы напоминать среду пользователя. Однако такая роскошь часто недоступна по ряду причин, и в подобных случаях комплексное тестирование системы является процессом контроля (т. е. выполняется в имитируемой, или тестовой среде). Например, в случае бортовой вычислительной системы космического корабля или системы противоракетной защиты вопрос о реальной среде (запуск настоящего космического корабля или выстрел настоящей ракетой) обычно не стоит. Кроме того, как мы увидим дальше, некоторые типы комплексных

тестов не осуществимы в реальной обстановке по экономическим соображениям, и лучше всего выполнять их в моделируемой среде.

Комплексное тестирование – наиболее творческий из всех видов тестирования. Разработка хороших комплексных тестов требует часто даже больше изобретательности, чем само проектирование системы. Здесь нет простых рекомендаций типа тестирования всех ветвей или построения функциональных диаграмм.

Комплексное тестирование системы – такая особая и такая важная работа, что в будущем возможно появление компаний, специализирующихся в основном на комплексном тестировании систем, разработанных другими.

По своей природе комплексные тесты никогда не сводятся к проверке отдельных функций системы. Они часто пишутся в форме сценариев, представляющих ряд последовательных действий пользователя. Например, один комплексный тест может представлять подключение терминала к системе, выдачу последовательно 10–20 команд и затем отключение от системы. Вследствие их особой сложности тесты системы состоят из нескольких компонентов: сценария, входных данных и ожидаемых выходных данных. В сценарии точно указываются действия, которые должны быть совершены во время выполнения теста.

1.2.4 Организация и этапы тестирования при испытаниях надежности сложных программных средств

Основные этапы тестирования и испытаний комплекса программ и его компонентов представлены на рисунке 1.2. Для каждого этапа на рисунке указаны основные исходные данные и результаты тестирования и испытаний. При тестировании, отладке и испытаниях корректности компонентов комплексов программ выделены следующие три этапа: 1) комплексирование модулей и отладка автономных групп программ в статике без взаимодействия с другими компонентами и, возможно, без подключения к операционной системе реального времени; 2) тестирование и отладка групп программ в статике с учетом взаимодействия с некоторыми другими важнейшими компонентами и с базой данных; 3) тестирование и отладка отдельных программных компонентов в реальном времени во взаимодействии с другими функциональными компонентами и с основными компонентами операционной системы и базы данных.

Сложность тестирования компонентов на этих этапах в значительной степени обусловлена несинхронным процессом их разработки и

отладки. Целесообразная последовательность отладки определенных компонентов может нарушаться неготовностью к сопряжению с ними других взаимодействующих программ. В результате к комплексному тестированию и испытаниям ПС могут быть готовы не все необходимые компоненты, и их приходится начинать с некоторой не всегда самой важной и целесообразной совокупности групп программ.

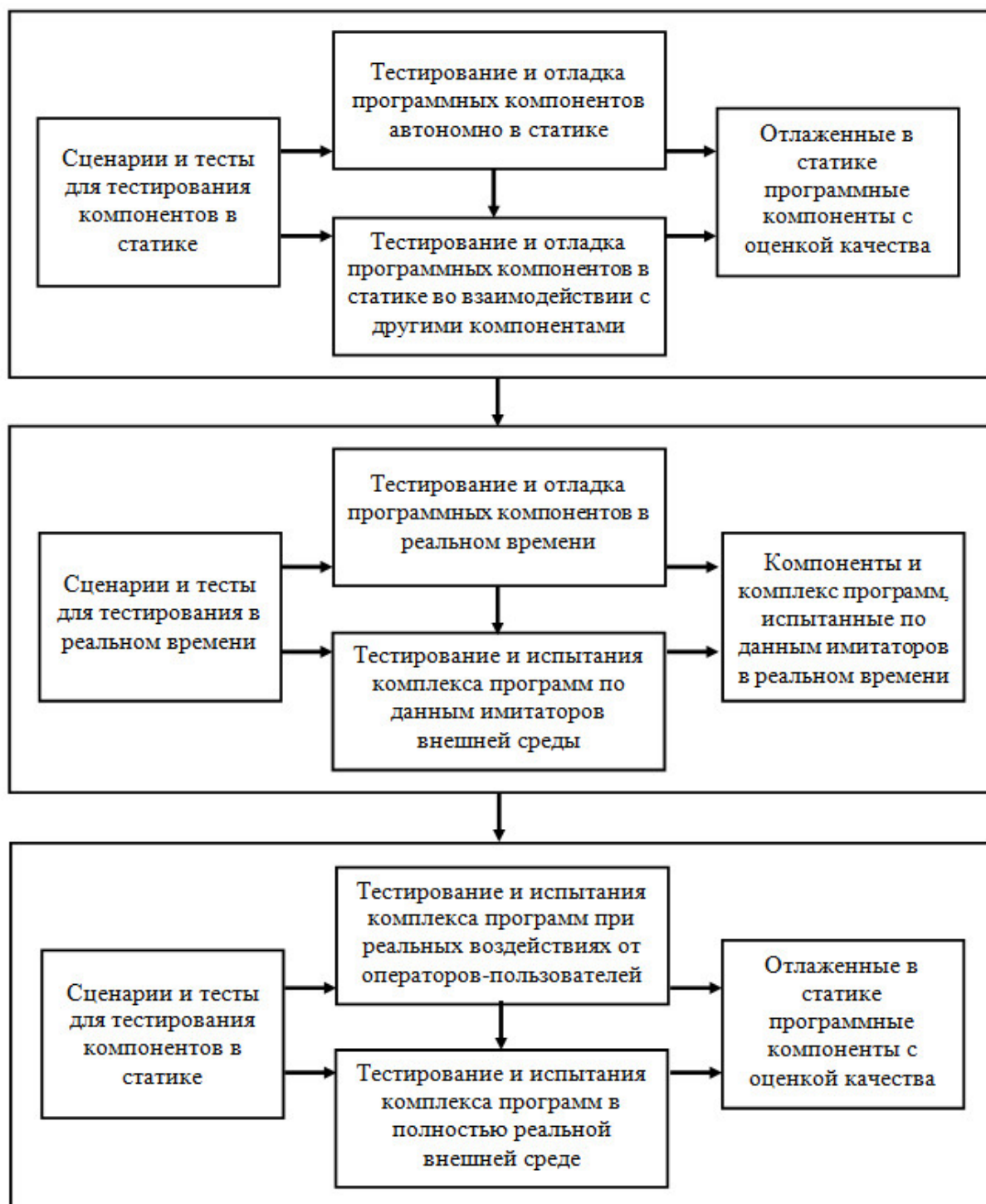


Рисунок 1.2 – Схема этапов тестирования и испытаний сложных комплексов программ

Для обеспечения имитации объектов внешней среды и других взаимодействующих групп программ на этих этапах используются частные генераторы соответствующих тестов. Эти генераторы тестов целесообразно разрабатывать и оформлять как отдельные модули или группы программ, функционирующие на той же технологической ЭВМ и в той же операционной среде, что и отлаживаемые компоненты. Совместно с ними также реализуются и функционируют частные специализированные программы для обработки отдельных результатов отладки соответствующих групп программ, что также требует некоторых ресурсов ЭВМ. При этом не всегда удастся полностью реализовать реальный масштаб времени для отдельных функциональных компонентов и приходится применять стартстопный режим тестирования или растягивать циклы решения функциональных задач и имитировать псевдореальный масштаб.

После комплексирования основных функциональных компонентов начинаются тестирование и испытания ПС в целом. Для них наиболее характерны следующие стадии комплексного тестирования и испытаний ПС в реальном времени:

- по данным моделирующего стенда или генераторов тестов, имитирующих отдельные объекты внешней среды;
- с имитаторами отдельных объектов внешней среды и с реальными воздействиями от операторов–пользователей;
- в полностью адекватной реальной или имитированной внешней среде и с реальными воздействиями от операторов–пользователей.

На всех стадиях отладки, кроме операций непосредственной проверки функционирования программ, можно выделить еще две важные группы работ. Первая группа – это работы по методическому обеспечению тестирования и по созданию средств автоматизированной генерации тестов. Вторая группа работ должна обеспечивать возможность обработки результатов тестирования и оценки достигнутых показателей качества функционирования программ.

Средства генерации тестов и обработки результатов отладки можно разделить на три вида (рисунок 1.2). Одни и те же средства автоматизации тестирования в статике обычно обеспечивают отладку групп программ как автономно, так и во взаимодействии с другими компонентами. Средства, имитирующие внешнюю среду в реальном времени, чаще всего ориентированы на отладку как функциональных компонентов, так и ПС в целом. Еще один вид генераторов тестов в той или иной степени использует реальные объекты внешней среды. Первоначально такими объектами являются имитирующие стенды с участием реального функционирования операторов–пользователей. Затем

источниками тестов могут быть полные комплексы реальной аппаратуры внешних объектов или их аппаратные аналоги.

Каждая из выделенных стадий тестирования может рассматриваться как обеспечивающая создание определенного промежуточного продукта – функциональной группы программ или программного средства с некоторыми ограниченными характеристиками качества. Эти характеристики выделяются и детализируются на основе первичного технического задания и спецификации требований на ПС. В процессе проектирования ПС они уточняются и конкретизируются в спецификациях требований на группы программ и их компоненты. В результате создается совокупность эталонов, имеющих последовательно расширяющиеся номенклатуру и наборы значений показателей качества, которым должны соответствовать отлаживаемые и испытываемые компоненты на каждой стадии тестирования.

Подобная совокупность эталонных показателей качества промежуточных продуктов обеспечивает возможность управления процессом тестирования с целью достижения необходимого качества конечного продукта – ПС при минимальных или разумных затратах. Для этого каждая стадия тестирования должна иметь фиксированный и документированный результат с определенными эталонами и достигнутыми характеристиками программ. Подобный систематизированный контроль тестирования гарантирует высокое качество ПС реального времени, к которым предъявляются обычно высокие требования по надежности. Кроме того, компоненты, прошедшие все стадии тестирования, с большой уверенностью могут применяться как повторно используемые компоненты в последующих версиях ПС.

Организация завершающих испытаний комплексов программ. Испытания главного конструктора, которые зачастую совмещаются с завершением комплексной отладки, должны оформляться документально и являются основанием для предъявления ПС заказчику на завершающиеся совместные испытания. Любые испытания ограничены допустимым объемом проверок и длительностью работы комиссии, поэтому не могут гарантировать абсолютную проверку изделия. Для повышения достоверности определения и улучшения характеристик ПС после испытаний главного конструктора программы целесообразно передавать некоторым пользователям на опытную эксплуатацию в типовых условиях. Это позволяет более глубоко оценить эксплуатационные характеристики созданного комплекса и устранить некоторые дефекты и ошибки. Опытная эксплуатация проводится разработчиками с участием испытателей и некоторых пользователей, назначаемых заказчиком. Результаты и показатели надежности опытной эксплуатации

после испытаний главного конструктора могут учитываться при проведении совместных испытаний для их сокращения.

Совместные приемо-сдаточные испытания проводятся комиссией заказчика, в которой участвуют главный конструктор разработки и некоторые ведущие разработчики, аттестованные сертификационной лабораторией. Комиссия при испытании руководствуется следующими документами: утвержденным заказчиком и согласованным с разработчиком техническим заданием и спецификациями на ПС; действующими государственными и ведомственными стандартами на проектирование и испытания программ и на техническую документацию, а также согласованными с заказчиком стандартами «де-факто»; программой испытаний по всем требованиям технического задания; методиками испытаний по каждому разделу требований технического задания; комплектом эксплуатационной документации на комплекс программ.

Программа испытаний является планом проведения серии экспериментов и разрабатывается с позиции минимизации объема тестирования в процессе проведения испытаний для проверки выполнения требований технического задания и соответствия предъявленной документации. Программа испытаний, методики их проведения и оценки результатов, разработанные совместно заказчиком и разработчиком, должны быть согласованы и утверждены. Они должны содержать уточнения и детализацию требований технического задания для данного ПС, а также гарантировать корректную проверку всех заданных характеристик, в том числе надежности. Программа испытаний должна содержать следующие четко сформулированные пять разделов: 1) **объект** испытаний, его назначение и перечень основных документов, определивших его разработку; 2) **цель** испытаний с указанием всех требований технического задания, подлежащих проверке, и ограничений на проведение испытаний; 3) собственно **программу** испытаний, содержащую проверку комплектности спроектированного ПС в соответствии с техническим заданием, и план тестирования для проверки по всем разделам технического задания и дополнительным требованиям, формализованным отдельными решениями разработчиков и заказчика; 4) **методики** испытаний, однозначно определяющие все понятия проверяемых характеристик, условия и сценарии тестирования, средства, используемые для испытаний; 5) методики **обработки** и оценки результатов тестирования по каждому разделу программы испытаний.

Большой объем разнородных данных, получаемых при испытаниях крупномасштабных ПС, и разнообразие возможных способов их обработки, интерпретации и оценки приводят к тому, что важнейшими

факторами становятся методики обработки и оценки результатов, а также протоколы проверки по пунктам программы испытаний. В соответствии с методиками испытаний средства автоматизации должны обеспечивать всю полноту проверок характеристик по каждому разделу методик. Результаты испытаний фиксируются в протоколах, которые обычно содержат следующие пять разделов: 1) назначение тестирования и раздел требований технического задания, по которому проводились испытания; 2) указания методик, в соответствии с которыми проводились испытания, обработка и оценка результатов; 3) условия и сценарии проведения тестирования и характеристики исходных данных; 4) обобщенные результаты испытаний с оценкой их на соответствие требованиям технического задания и другим руководящим документам, а также технической документации; 5) выводы о результатах испытаний и соответствии созданного ПС определенному разделу требований технического задания.

Протоколы по всей программе испытаний обобщаются в акте, в результате чего делается заключение о соответствии системы требованиям заказчика и завершении работы с положительным или отрицательным итогом. При полном выполнении всех требований технического задания заказчик обязан принять систему, и работа считается завершенной.

Наиболее полным и разносторонним испытаниям должна подвергаться первая базовая версия ПС. При испытаниях очередных модернизированных версий ПС возможны значительные сокращения объемов тестирования повторно используемых компонентов. Однако комплексные и завершающие испытания каждой новой версии ПС, как правило, проводятся в полном объеме, гарантирующем проверку выполнения всех требований измененного технического задания. Для обеспечения выявления дефектов в процессе эксплуатации серийных образцов в каждом из них должен быть предусмотрен некоторый минимум средств проверки функционирования и обнаружения искажений результатов. Этот минимум средств должен позволять фиксировать условия неправильной работы программ и характер проявления дефектов. Последующее исправление ошибок должно проводиться специалистами, осуществляющими сопровождение.

При завершающих испытаниях основное внимание, кроме проверок функциональной пригодности, должно сосредотачиваться на подготовке стрессовых тестов, тестировании в режимах предельного использования ресурсов, надежности функционирования ПС. Задача испытателей и заказчика при проведении совместных испытаний состоит в выделении условий и области изменения переменных, которые

недостаточно проверены разработчиком и важны для последующего надежного функционирования программ. При этом разработчик контролирует, чтобы планируемые сценарии и тесты не выходили из областей, заданных техническим заданием и спецификацией требований. Испытания за пределами технического задания могут квалифицироваться как его расширение или могут исключаться по требованию разработчика.

До начала испытаний подлежат проверке и паспортизации средства, обеспечивающие получение эталонных данных, средства имитации тестов от внешних объектов, средства фиксирования и обработки результатов тестирования. При испытаниях важную роль играют оценка и обеспечение близких значений методической и статистической достоверности результатов испытаний. Методическая достоверность приемосдаточных испытаний ПС определяется следующими факторами:

- полнотой программы испытаний и корректностью методик тестирования по охвату возможных условий и сценариев функционирования программ и областей изменения исходных данных;
- достоверностью и точностью эталонных значений, с которыми сравниваются результаты тестирования испытываемой программы или которые служат опорными при расчете параметров, зафиксированных в техническом задании;
- адекватностью и точностью моделей, используемых для имитации тестов от внешней среды и подыгрыша их реакции на управляющие воздействия;
- точностью и корректностью регистрации и обработки результатов тестирования, а также сравнения полученных данных с требованиями технического задания.

Представленная выше организация испытаний сложных ПС ориентирована на наличие конкретного заказчика комплекса программ и ограниченное число пользователей, контролируемых заказчиком. Несколько иначе организуются испытания коммерческих пакетов прикладных программ, создаваемых по инициативе разработчиков для широкого круга пользователей при отсутствии конкретного заказчика. Для таких коммерческих прикладных программ принято проводить испытания в два последовательных этапа – Альфа- и Бета-тестирование. Испытания проводятся на соответствие критериям, формализованным руководителем проекта. Они заключаются в нормальной и форсированной (стрессовой) опытной эксплуатации конечными пользователями оформленного программного продукта в соответствии с сопроводительной документацией и различаются количеством участвующих пользователей.

При Альфа-тестировании привлекаются конечные пользователи, работающие в той же компании, но не участвовавшие непосредственно в разработке комплекса программ. Для Бета-тестирования привлекаются добровольные пользователи (потенциальные покупатели), которым бесплатно передается версия ПС для опытной эксплуатации. При этом особое значение имеет выделение компетентных, тщательных и доброжелательных пользователей, способных своими рекомендациями улучшить качество испытываемых программ. Их деятельность стимулируется бесплатным и ранним получением и освоением нового программного продукта и собственной оценкой его качества. Эти пользователи обязуются сообщать разработчикам сведения о всех выявленных дефектах и ошибках, а также вносить изменения в программы и данные или заменять версии по указаниям разработчиков. Только после успешной эксплуатации и Бета-тестирования ограниченным контингентом пользователей, руководителем проекта или фирмы разработчиков принимается решение о передаче ПС в продажу для широкого круга пользователей. Обобщение результатов Бета-тестирования может использоваться как часть или основа сертификационных испытаний.

При Альфа- и Бета-испытаниях принято разделять прогрессивное и регрессивное тестирование. Под прогрессивным понимается тестирование новых программных компонентов, для выявления дефектов и ошибок в исходных текстах программ и спецификациях. Регрессивное тестирование предназначено для контроля качества и корректности изменений в программах и данных после проведения корректировок. Необходимость и широта регрессивного тестирования определяются тем, что значительная доля изменений после Альфа- и Бета-тестирования, в свою очередь, содержит ошибки. Объем тестов и длительность обоих этапов тестирования определяются руководителями проекта в зависимости от сложности комплекса программ и интенсивности потока изменений.

«Вопросы и задания для самоконтроля»

1. Перечислите этапы или виды деятельности при тестировании ПО. Назовите наиболее важный этап.
2. Какова схема спектра подходов к проектированию тестов? Что общего в ней со стратегиями тестирования?
3. Исходя из какого соображения может формироваться набор тестов при тестировании по отношению к тексту программы?
4. Каковы причины мотивации тестирования программы в целом начинать с тестирования модулей?

- 5 На какую стратегию ориентировано тестирование модулей?
- 6 Назовите основные подходы в тестировании при слиянии модулей.
- 7 Каковы достоинства и недостатки нисходящего метода тестирования?
- 8 Какой подход в тестировании при слиянии модулей самый распространенный? Когда он неприменим?
- 9 В чем достоинства и недостатки пошагового тестирования?
- 10 Нисходящее тестирование и нисходящая разработка – это синонимы?
- 11 Восходящее тестирование и восходящая разработка – это синонимы?
- 12 Что такое комплексное тестирование?
- 13 Что такое тестирование внешних функций и чем оно отличается от комплексного тестирования?
- 14 Всегда ли осуществимо комплексное тестирование?
- 15 Приведите пример содержательного описания комплексного теста.
- 16 Назовите разновидности средств генерации тестов и обработки результатов отладки.
- 17 Назовите существующие варианты схем организации испытаний сложных программных систем.
- 18 Какова структура программы испытаний (разделы) для комплексного тестирования?
- 19 Какова структура программы обработки результатов испытаний (разделы) для комплексного тестирования?
- 20 Перечислите факторы, характеризующие достоверность приемосдаточных испытаний ПС.
- 21 Дайте характеристику тестов (данных, используемых при тестировании) и субъектов (кто проводит тестирование и что имеет за работу), проводящих Альфа- и Бета-тестирование. Изобразите схематично в виде таблицы.
- 22 Что понимается под прогрессивным и регрессивным тестированием?

1.3 Тестирование программного обеспечения

1.3.1 Тестирование программного обеспечения

На современном этапе развития информационных технологий ПО характеризуется большой степенью сложности. Особенностью ПО,

разрабатываемого для сферы экономики, является то, что оно постоянно изменяется. Связано это прежде всего с изменениями, происходящими в предметной области (введение новых услуг, бизнес-процессов, изменение законодательства).

Создание и поддержка банка тестов – сложная задача и требует высокой квалификации сотрудников отдела тестирования. За все надо платить, но качество конечного продукта того стоит. Тестирование – это дорогостоящий и трудоемкий процесс, поэтому зарубежными компаниями ведутся разработки в области автоматизации тестирования. Попытки применения автоматизации тестирования связаны с тем, что в принципе невозможно полностью протестировать программный продукт, соответственно специализированные пакеты приближают «покрытие» тестами программы к 100 %. На рынке специальных сред для тестирования программного обеспечения можно отметить разработки ведущих в этой области фирм: Rational (Visual Test, Rational Robot, Team Test и др.), Mercury Interactive (WinRunner), Segue Software (QA Partner).

Сегодня можно и нужно говорить о программном обеспечении как о промышленном продукте, соответственно о создании ПО – как о производстве. Существует множество стандартов поддержки жизненного цикла программного обеспечения. Разработано множество стандартов и методик поддержки стадий ЖЦ ПО, например стандарты ISO 9000 и ISO 9001, разработанные Международной организацией по стандартизации (ISO).

1.3.2 Место и цель этапа тестирования программного обеспечения

На сегодняшний день автоматизировано большинство этапов разработки программного обеспечения, в том числе и этап тестирования. Однако в отечественной практике тестированию программных средств отведена незаслуженно маленькая роль. Причинами могут быть отсутствие денег на приобретение дорогостоящих CASE-средств, поддерживающих все этапы разработки ПО, в том числе тестирование, или нежелание держать такую штатную единицу, как специалист по тестированию. Обычно приобретают средства автоматизации проектирования (создание ER-моделей, информационных моделей и пр.) и программирования (автоматическое создание БД на основе ER-модели, создание интерфейса на основе ER-модели, визуальное программирование). С тестированием дела обстоят сложнее.

Тестирование занимает важное место в жизненном цикле программного обеспечения, это трудоемкий и дорогостоящий процесс. В организационной структуре современной фирмы – разработчика ПО должен быть отдел по тестированию программного обеспечения или специалист по тестированию программного обеспечения, так как одним из принципов тестирования является избежание тестирования автором, а тестирование программного средства напрямую связано с его качеством.

Отечественные производители коммерческого программного обеспечения только сейчас серьезно задумались о качестве своей продукции и, как следствие, о тестировании. В качестве объективных причин, почему это происходит именно сейчас и почему этого не было раньше, можно выделить следующие.

1 Сформировалась нормативно-правовая база для осуществления деятельности по разработке программного обеспечения, база в области авторского права и смежных прав, а также защиты прав потребителей.

2 Законодательно закрепленные принципы работают и активно используются в правоприминительной практике. Особенно это относится к защите авторских прав.

3 Исполнительная власть начала пресекать попытки распространения нелегального программного обеспечения. Есть прецеденты процессов, по которым возмещаются потери от нарушения авторских прав.

4 Возросли требования заказчика к качеству программного обеспечения. Данный момент связан с правовой базой (законы о защите прав потребителей), жесткой конкуренцией на рынке.

5 Накоплен большой опыт создания программных средств, российские компании выходят на другие рынки (в том числе на мировой рынок), что влечет необходимость выполнения новых норм по качеству программных средств.

По этим причинам процессы обеспечения и контроля качества, одним из которых является тестирование, приобретают в настоящий момент большое значение и актуальность. Целью тестирования является обнаружение максимального количества ошибок, а не всех ошибок в программе. Обнаружение всех ошибок невозможно. Полное и абсолютное тестирование выглядит скорее мечтой, чем реальностью.

В 1979 г. Майерс (Myers) описал некоторый простой алгоритм, в котором был один цикл и несколько операторов условного перехода. В большинстве языков программирования для кодирования такого алгоритма потребуется не более 20 строк кода. Но такая программа имеет более 100 триллионов путей выполнения! Самому быстрому

тестировщику для полного тестирования потребовался бы как минимум миллион лет. Аналогичная программа из 100 строк имеет 1018 триллионов путей выполнения, если на проверку выполнения одного пути тратить 1 секунду, то на полное ее завершение не хватит времени существования всей нашей вселенной, время жизни которой меньше $4 \cdot 10^{17}$ секунд!

Таким образом, целью тестирования является не тотальное обнаружение всех ошибок (это принципиально невозможно), а выявление наибольшего количества наиболее критичных ошибок. Если исправление их задерживается, то пользователи программного продукта должны быть предупреждены о наличии такого рода ошибок и рекомендуемых путях обхода.

Если процесс тестирования становится бесконечным, а полное тестирование невозможно, то как принять решение о выпуске в свет исследуемой версии программного продукта? Основными критериями завершенности тестирования является отсутствие критичных ошибок, каждая из которых может сделать абсолютно невозможной реализацию декларированной в системе прикладной функциональности (решение принимается по результатам функционального тестирования). Кроме того, при принятии решения учитывается общее количество зарегистрированных, но неисправленных ошибок. Компания–разработчик обычно заранее выбирает по каждому программному продукту общее количество ошибок (лимит), с которым уже нельзя выпускать программный продукт.

Количественная оценка завершенности процесса тестирования и готовности программного продукта для эксплуатации может быть получена при помощи моделей надежности ПО. Тестирование программного обеспечения имеет тесную связь с *качеством ПО*.

Самый простой способ представления информации для принятия решения – графический: по одной оси откладывается время от начала процесса тестирования, по другой – количество обнаруженных ошибок в программном средстве. По графику (знаку производной) определяется необходимость продолжения тестирования. Существует множество методов, которые помогают принять решение в выпуске программного обеспечения, однако самое веское слово остается за специалистом, осуществляющим тестирование программного обеспечения, так как на основе количества и характера найденных проблем он может судить о том, удовлетворит данный продукт потребности и ожидания пользователя или нет.

Современные идеологи проблем качества разделяют понятие «качество» на *внешнее* (external) и *внутреннее* (internal). Внешнее качество программного обеспечения – его способность удовлетворить

потребность конечного пользователя. Именно на это и направлен процесс тестирования программного обеспечения – обнаружение ошибок и несоответствий, т. е. в процессе тестирования выявляются те моменты (ошибки, неправильная реализация или отсутствие функциональных возможностей), которые не удовлетворили бы конечного пользователя. Тестирование программного обеспечения обеспечивает контроль качества продукта, поставляемого конечным пользователям.

Внутреннее качество программного обеспечения связано с удобством его производства для тех, кто его производит, с его технологичностью, стандартизованностью, безопасностью. Вопросы внутреннего качества в большей степени связаны с реализацией процессов жизненного цикла программного обеспечения, с процессами управления разработкой программного обеспечения. Улучшая документированность тестов, их более простую адаптируемость от версии к версии программного обеспечения, специалист по тестированию улучшает внутреннее качество программного обеспечения.

1.3.3 Виды тестирования

Наиболее важные виды тестирования программных средств перечислены ниже.

1 Функциональное – тестирование возможностей системы, ее реакция на те или иные ситуации. Обычно результат тестирования (реакция системы) сравнивается с постановкой задачи, при несоответствии фиксируется ошибка.

2 Регрессионное – проверка полноты реализуемых функций системы по сравнению с предыдущей версией программного продукта.

3 Нагрузочное – тестирование работы системы на пиковую нагрузку, при этом делается вывод о производительности системы. Например, выясняется среднее время ввода одного документа (если программное обеспечение предназначено для хранения и обработки документов). Условием для нагрузочного тестирования является выполнение испытаний на одной и той же конфигурации системы. Если тестируется производительность на 2-х разных СУБД, то конфигурация системы должна быть идентичной (тот же сервер, те же рабочие станции), в испытаниях меняются лишь СУБД. На основе нагрузочного тестирования выдвигаются требования к аппаратной части и программной части системы (операционная система, СУБД).

4 Контроль после исправления (обратная связь). Этот вид тестирования подразумевает под собой проверку уже исправленных ошибок.

5 Стрессовое тестирование – проверка реакции системы на внештатные ситуации. Примером может служить проверка системы на восстановление работоспособности после отключения питания на сервере базы данных.

6 Адаптационное тестирование – проверка корректности перевода программного обеспечения на другой национальный язык.

13.4 Передовые технологии в тестировании (автоматизация тестирования)

При контроле качества, лучшие результаты дает использование автоматических тестов с применением специальных промышленных средств автоматизации тестирования. Высокую эффективность имеют также специальные тестовые процедуры, написанные на языке программирования, на котором написано само ПС. Преимущества использования автоматических тестов перед тестированием очевидны: они беспристрастны; позволяют выполнять проверку необходимое количество раз; не устают и не ошибаются; могут протестировать гораздо больше за меньшее количество времени; не требуют дополнительной оплаты при работе по ночам и выходным; более четко отвечают на вопрос, что протестировано и с каким результатом.

Создание и поддержка банка тестов сложная задача, требующая высокой квалификации сотрудников отдела тестирования. Автоматизация тестирования связана с тем, что в принципе невозможно полностью протестировать программный продукт, соответственно специализированные пакеты приближают «покрытие» тестами программы к 100 %. На рынке специальных сред для тестирования программного обеспечения можно отметить разработки ведущих в этой области фирм: Rational (Visual Test, Rational Robot, Team Test и др.), Mercury Interactive (WinRunner), Segue Software (QA Partner). Такое ПО весьма специфично и имеет достаточно высокую цену – порядка нескольких десятков тысяч долларов. Пакеты тестирования можно разделить по поддерживаемой стратегии тестирования на пакеты, поддерживающие стратегию «белого ящика» и на поддерживающие стратегию «черного ящика».

Пакеты, реализующие стратегию «белого ящика», позволяют: записывать, а потом воспроизводить последовательность пользовательского ввода (нажатие клавиатуры, движения «мышью»); распознавать объекты и их свойства (окна Windows, текст в окне и пр.); запоминать копию экрана; сравнивать состояние программы относительно предыдущего

тестового прогона; производить математические вычисления на основе данных из тестируемой программы; замерять выполнение одной и той же последовательности действий в различных условиях; эмулировать выполнение программы несколькими пользователями одновременно; записывать подробный протокол выполнения автоматического теста; другие функции.

Пакеты, реализующие стратегию «черного ящика», позволяют: отслеживать выполнение того или иного фрагмента кода программы; подсчитывать количество выполнения того или иного фрагмента кода программы; вычислять время выполнения участка кода (важно при пересмотрах кода и его оптимизации); подсчитывать общее «покрытие» программы; автоматически контролировать значение переменных и выдавать ошибку или предупреждение, если значения не совпадают с теми, которые ожидаются; на основе данных, полученных от пакета автоматизации тестирования, возможно выполнять расчеты о надежности программного обеспечения; получать различные статистические данные о программе.

Средства автоматизации тестирования не предполагают отсутствие инженера по тестированию, а требуют от него новых знаний. Программа автоматизации тестов не выполнит всю работу по тестированию сама. Для нее нужны специальные инструкции – сценарии тестов, написанные на специально разработанном языке. Таким образом, автоматизация заключается в избавлении инженера по тестированию от рутинной работы, теперь тестер занимается разработкой тестов и программированием тестов на языке системы автоматизации тестирования.

Вопросы и задания для самоконтроля

- 1 Каковы объективные причины усиления внимания к процессу тестирования?
- 2 Назовите основные критерии завершенности тестирования
- 3 В чем проявляется внешнее качество программного обеспечения?
- 4 В чем проявляется внутреннее качество программного обеспечения?
- 5 Назовите наиболее важные виды тестирования ПС.
- 6 Дайте краткую характеристику функционального, регрессионного, нагрузочного, стрессового и адаптационного тестирований, а также контроля после исправления.
- 7 Каковы преимущества автоматических средств тестирования?
- 8 Как систематизируются автоматические средства тестирования?

9 В чем функциональность автоматических средств тестирования, реализующих стратегию «белого ящика»?

10 В чем функциональность автоматических средств тестирования, реализующих стратегию «черного ящика»?

11 Каковы задачи тестера при использовании средств автоматизации?

1.4 Виды тестирования программного обеспечения

1.4.1 Функциональные виды тестирования

Все виды тестирования программного обеспечения, в зависимости от преследуемых целей, можно условно разделить на следующие группы: 1) функциональные; 2) нефункциональные; 3) связанные с изменениями.

Функциональные тесты базируются на функциях и особенностях, а также взаимодействии с другими системами, и могут быть представлены на всех уровнях тестирования: компонентном или модульном (Component/Unit testing), интеграционном (Integration testing), системном (System testing) и приемочном (Acceptance testing). Функциональные виды тестирования рассматривают внешнее поведение системы. Далее перечислены самые распространенные виды функциональных тестов:

- функциональное тестирование (Functional testing);
- тестирование безопасности (Security and Access Control Testing);
- тестирование взаимодействия (Interoperability Testing).

Функциональное тестирование. Этот вид тестирования проверяет соответствие реализованных функций требованиям, техническому заданию, спецификациям, различным другим проектным документам и просто ожиданиям пользователя. Проверяется каждая из функций приложения и все они в комплексе. Исследуются все сценарии использования. Проверяется адекватность хранимых и выходных данных, методы их обработки, обработка вводимых данных, методы хранения данных, методы импорта и экспорта данных и т. д. в зависимости от специфики приложения.

Функциональные тесты основываются на функциях, выполняемых системой, и могут проводиться на всех уровнях тестирования (компонентном, интеграционном, системном, приемочном). Как правило, эти функции описываются в требованиях, функциональных спецификациях или в виде случаев использования системы (use cases).

Тестирование функциональности может проводиться в двух аспектах: «требования»; «бизнес-процессы».

Тестирование в перспективе «требования» использует спецификацию функциональных требований к системе как основу для дизайна тестовых случаев (Test Cases). В этом случае необходимо сделать список того, что будет тестироваться, а что нет, приоритезировать требования на основе рисков (если это не сделано в документе с требованиями), а на основе этого приоритезировать тестовые сценарии (test cases). Это позволит сфокусироваться и не упустить при тестировании наиболее важный функционал.

Тестирование в перспективе «бизнес-процессы» использует знание этих самых бизнес-процессов, которые описывают сценарии ежедневного использования системы. В этой перспективе тестовые сценарии (test scripts), как правило, основываются на случаях использования системы (use cases).

Преимущества функционального тестирования: имитирует фактическое использование системы. Недостатки функционального тестирования: возможность упущения логических ошибок в программном обеспечении; вероятность избыточного тестирования.

Достаточно распространенной является автоматизация функционального тестирования.

Тестирование безопасности. Стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным. Тестирование безопасности может выполняться как автоматизированно так и в ручную, включая проверку как позитивных, так и негативных тестовых случаев. Основывается на трех основных принципах – это конфиденциальность, целостность и доступность (confidentiality, integrity, availability)

Конфиденциальность – это сокрытие определенных ресурсов или информации. Под конфиденциальностью можно понимать ограничение доступа к ресурсу некоторой категории пользователей, или другими словами, при каких условиях пользователь авторизован получить доступ к данному ресурсу.

Существует два основных критерия при определении понятия целостности:

1) доверие; ожидается, что ресурс будет изменен только соответствующим способом определенной группой пользователей;

2) повреждение и восстановление; в случае, когда данные повреждаются или неправильно меняются авторизованным или

не авторизованным пользователем, необходимо определить, насколько важной является процедура восстановления данных.

Доступность представляет собой требования о том, что ресурсы должны быть доступны авторизованному пользователю, внутреннему объекту или устройству. Как правило, чем более критичен ресурс, тем выше уровень доступности должен быть.

Тестирование взаимодействия. С развитием сетевых технологий и интернета взаимодействие разных систем, сервисов и приложений друг с другом приобрело значительную актуальность, так как любые связанные с этим проблемы могут привести к падению авторитета компании, что, как следствие, повлечет за собой финансовые потери. Поэтому к тестированию взаимодействия стоит подходить со всей серьезностью.

Тестирование взаимодействия – это функциональное тестирование, проверяющее способность приложения взаимодействовать с одним и более компонентами или системами и включающее в себя тестирование совместимости (compatibility testing) и интеграционное тестирование (integration testing).

Программное обеспечение с хорошими характеристиками взаимодействия может быть легко интегрировано с другими системами, не требуя каких-либо серьезных модификаций. В этом случае количество изменений и время, требуемое на их выполнение, могут быть использованы для измерения возможности взаимодействия.

1.4.2 Нефункциональные виды тестирования

Нефункциональное тестирование описывает тесты, необходимые для определения характеристик программного обеспечения, которые могут быть измерены различными величинами. В целом, это тестирование того, «Как» система работает. Далее перечислены основные виды нефункциональных тестов:

- тестирование производительности:
 - а) нагрузочное тестирование (Performance and Load Testing);
 - б) стрессовое тестирование (Stress Testing);
 - в) тестирование стабильности или надежности (Stability / Reliability Testing);
 - г) объемное тестирование (Volume Testing);
- тестирование установки (Installation testing);
- тестирование удобства пользования (Usability Testing) (см. 1.4.4);
- тестирование на отказ и восстановление (Failover and Recovery Testing) (см. 1.4.5);

– конфигурационное тестирование (Configuration Testing) (см. 1.4.6).

При *нагрузочном тестировании* тестирования производительности определяются возможности приложения под нагрузкой, при этом происходит: измерение времени выполнения выбранных операций при определенных 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 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.

Задачей *объемного тестирования* является получение оценки производительности при увеличении объемов данных в базе данных приложения, при этом происходит: измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций; может производиться определение количества пользователей, одновременно работающих с приложением.

Задачей *тестирования стабильности* (надежности) является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки. Времена выполнения операций могут 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.

1.4.3 Связанные с изменениями виды тестирования

После проведения необходимых изменений, таких как исправление бага/дефекта, программное обеспечение должно быть перетестировано для подтверждения того факта, что проблема была действительно решена. Ниже перечислены виды тестирования, которые необходимо

проводить после установки программного обеспечения, для подтверждения работоспособности приложения или правильности осуществленного исправления дефекта:

- дымовое тестирование (Smoke Testing);
- регрессионное тестирование (Regression Testing);
- тестирование сборки (Build Verification Test);
- санитарное тестирование или проверка согласованности / исправности (Sanity Testing).

Понятие *дымовое тестирование* появилось в инженерной среде. При вводе в эксплуатацию нового оборудования («железа») считалось, что тестирование прошло удачно, если из установки не пошел дым. В области же тестирования программного обеспечения, оно направлено на поверхностную проверку всех модулей приложения на предмет работоспособности и наличие быстро находимых критических и блокирующих дефектов. По результатам дымового тестирования делается вывод о том, принимается или нет установленная версия программного обеспечения в тестирование, эксплуатацию или на поставку заказчику. Для облегчения работы, экономии времени и людских ресурсов рекомендуется внедрить автоматизацию тестовых сценариев для дымового тестирования.

Регрессионное тестирование – это вид тестирования, направленный на проверку изменений, сделанных в приложении или окружающей среде (починка дефекта, слияние кода, миграция на другую операционную систему, базу данных, веб сервер или сервер приложения), для подтверждения того факта, что существующая ранее функциональность работает как и прежде (см. также Санитарное тестирование или проверка согласованности / исправности). Регрессионными могут быть как функциональные, так и нефункциональные тесты.

Как правило, для регрессионного тестирования используются тест-кейсы, написанные на ранних стадиях разработки и тестирования. Это дает гарантию того, что изменения в новой версии приложения не повредили уже существующую функциональность. Рекомендуется делать автоматизацию регрессионных тестов для ускорения последующего процесса тестирования и обнаружения дефектов на ранних стадиях разработки программного обеспечения.

Сам по себе термин «Регрессионное тестирование» в зависимости от контекста использования может иметь разный смысл. Например, выделяют три основных типа регрессионного тестирования: регрессия багов (Bug regression) – попытка доказать, что исправленная ошибка на самом деле не исправлена; регрессия старых багов (Old bugs regression) – попытка доказать, что недавнее изменение кода или

данных сломало исправление старых ошибок, т. е. старые баги стали снова воспроизводиться; регрессия побочного эффекта (Side effect regression) – попытка доказать, что недавнее изменение кода или данных сломало другие части разрабатываемого приложения.

Санитарное тестирование или проверка согласованности / исправности (Sanity Testing) – это узконаправленное тестирование, достаточное для доказательства того, что конкретная функция работает согласно заявленным в спецификации требованиям. Является подмножеством регрессионного тестирования. Используется для определения работоспособности определенной части приложения после изменений, произведенных в ней или окружающей среде. Обычно выполняется вручную.

Отличие санитарного тестирования от дымового. В некоторых источниках ошибочно полагают, что санитарное и дымовое тестирование – это одно и то же. Мы же полагаем, что эти виды тестирования имеют «вектора движения», направления в разные стороны. В отличие от дымового (Smoke testing), санитарное тестирование (Sanity testing) направлено вглубь проверяемой функции, в то время как дымовое направлено вширь, для покрытия тестами как можно большего функционала в кратчайшие сроки.

Тестирование сборки (Build Verification Test) – это тестирование, направленное на определение соответствия, выпущенной версии, критериям качества для начала тестирования. По своим целям является аналогом Дымового Тестирования, направленного на приемку новой версии в дальнейшее тестирование или эксплуатацию. Вглубь оно может проникать дальше, в зависимости от требований к качеству выпущенной версии.

Тестирование установки (Installation Testing) – направлено на проверку успешной инсталляции и настройки, а также обновления или удаления программного обеспечения. В настоящий момент наиболее распространена установка ПО при помощи инсталляторов (специальных программ, которые сами по себе также требуют надлежащего тестирования). В реальных условиях инсталляторов может не быть. В этом случае придется самостоятельно выполнять установку программного обеспечения, используя документацию в виде инструкций или readme файлов, шаг за шагом описывающих все необходимые действия и проверки. В распределенных системах, где приложение разворачивается на уже работающем окружении, простого набора инструкций может быть мало. Для этого зачастую пишется план установки (Deployment Plan), включающий не только шаги по инсталляции приложения, но и шаги отката (roll-back) к предыдущей

версии, в случае неудачи. Сам по себе план установки также должен пройти процедуру тестирования для избежания проблем при выдаче в реальную эксплуатацию. Особенно это актуально, если установка выполняется на системы, где каждая минута простоя – это потеря репутации и большого количества средств, например: банки, финансовые компании или даже баннерные сети. Поэтому тестирование установки можно назвать одной из важнейших задач по обеспечению качества программного обеспечения.

Именно такой комплексный подход с написанием планов, пошаговой проверкой установки и отката инсталляции, полноправно можно назвать тестированием установки или Installation Testing.

1.4.4 Тестирование удобства пользования

Тестирование удобства пользования (Usability Testing). Иногда мы сталкиваемся с непонятными, нелогичными приложениями, многие функции и способы использования которых часто не очевидны. После такой работы редко возникает желание использовать приложение снова, и мы ищем более удобные аналоги. Для того чтобы приложение было популярным, ему мало быть функциональным – оно должно быть еще и удобным. Если задуматься, интуитивно понятные приложения экономят нервы пользователям и затраты работодателя на обучение. А значит, они более конкурентоспособные! Поэтому тестирование удобства использования, о котором пойдет речь далее, является неотъемлемой частью тестирования любых массовых продуктов.

Тестирование удобства пользования – это метод тестирования, направленный на установление степени удобства использования, обучаемости, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий. [ISO 9126]

Тестирование удобства пользования дает оценку уровня удобства использования приложения по следующим пунктам:

- производительность, эффективность (efficiency) – сколько времени и шагов понадобится пользователю для завершения основных задач приложения, например, размещение новости, регистрации, покупка и т. д.? (меньше – лучше);
- правильность (accuracy) – сколько ошибок сделал пользователь во время работы с приложением? (меньше – лучше);
- активизация в памяти (recall) – как много пользователь помнит о работе приложения после приостановки работы с ним на длительный период времени? (повторное выполнение операций после прерыва должно проходить быстрее чем у нового пользователя);

– эмоциональная реакция (emotional response) – как пользователь себя чувствует после завершения задачи – растерян, испытал стресс? Посоветует ли пользователь систему своим друзьям? (положительная реакция – лучше).

Уровни проведения. Проверка удобства использования может проводиться как по отношению к готовому продукту, посредством тестирования черного ящика (black box testing), так и к интерфейсам приложения (API), используемым при разработке – тестирование белого ящика (white box testing). В этом случае проверяется удобство использования внутренних объектов, классов, методов и переменных, а также рассматривается удобство изменения, расширения системы и интеграции ее с другими модулями или системами. Использование удобных интерфейсов (API) может улучшить качество, увеличить скорость написания и поддержки разрабатываемого кода, и, как следствие, улучшить качество продукта в целом.

Отсюда – становится очевидным, что тестирование удобства пользования может производиться на разных уровнях разработки программного обеспечения: модульном, интеграционном, системном и приемочном. При этом оно целиком и полностью зависит от того, кто будет использовать приложение на выделенном конкретном уровне – разработчик, бизнес-пользователь системы и т. д.

Заблуждения о тестировании удобства пользования.

1. *Тестирование пользовательского интерфейса = Тестирование удобства пользования.* Тестирование удобства пользования не имеет ничего общего с тестированием функциональности пользовательского интерфейса, оно лишь проводится на пользовательском интерфейсе, равно как и на многих других возможных компонентах продукта. При этом тип тестирования и тесткейсы будут совсем другие, так как речь может идти об удобстве использования не визуальных компонентов (если таковые имеются) или процессе администрирования, например, распределенного клиент-серверного продукта и т. д.

2. *Тестирование удобства пользования можно провести без участия эксперта.* Не всегда человек, не разбирающийся в предметной области, способен провести его самостоятельно. Представьте, что тестировщику нужно протестировать удобство пользования стратегического бомбардировщика. Ему придется проверить основные функции: удобство ведения боя, навигации, пилотирования, обслуживания, наземной транспортировки и т. д. Очевидно, что без привлечения эксперта это будет весьма проблематично, и можно даже сказать, невозможно.

1.4.5 Тестирование на отказ и восстановление

Тестирование на отказ и восстановление (Failover and Recovery Testing) проверяет тестируемый продукт с точки зрения способности противостоять и успешно восстанавливаться после возможных сбоев, возникших в связи с ошибками программного обеспечения, отказами оборудования или проблемами связи (например, отказ сети). Целью данного вида тестирования является проверка систем восстановления (или дублирующих основной функционал систем), которые, в случае возникновения сбоев, обеспечат сохранность и целостность данных тестируемого продукта.

Тестирование на отказ и восстановление очень важны для систем, работающих по принципу «24x7». Если создается продукт, который будет работать, например, в интернете, то без проведения данного вида тестирования просто не обойтись. Так как каждая минута простоя или потеря данных в случае отказа оборудования, может стоить денег, потери клиентов и репутации на рынке.

Методика подобного тестирования заключается в симулировании различных условий сбоя и последующем изучении и оценке реакции защитных систем. В процессе подобных проверок выясняется, была ли достигнута требуемая степень восстановления системы после возникновения сбоя.

При достижении соответствующих условий сбоя и по результатам работы систем восстановления можно оценить продукт с точки зрения тестирования на отказ. Во всех вышеперечисленных случаях по завершении процедур восстановления должно быть достигнуто определенное требуемое состояние данных продукта: потеря или порча данных в допустимых пределах; отчет или система отчетов с указанием процессов или транзакций, которые не были завершены в результате сбоя.

1.4.6 Конфигурационное тестирование

Конфигурационное тестирование (Configuration Testing) – специальный вид тестирования, направленный на проверку работы программного обеспечения при различных конфигурациях системы (заявленных платформах, поддерживаемых драйверах, при различных конфигурациях компьютеров и т. д.)

В зависимости от типа проекта конфигурационное тестирование может иметь разные цели: 1) для проекта по профилированию работы системы цель тестирования: определить оптимальную конфигурацию

оборудования, обеспечивающую требуемые характеристики производительности и времени реакции тестируемой системы; 2) для проекта по миграции системы с одной платформы на другую цель тестирования: проверить объект тестирования на совместимость с объявленным в спецификации оборудованием, операционными системами и программными продуктами третьих фирм.

Уровни проведения тестирования. Для клиент-серверных приложений конфигурационное тестирование можно условно разделить на два уровня (для некоторых типов приложений может быть актуален только один): серверный или клиентский. На первом (серверном) уровне тестируется взаимодействие выпускаемого программного обеспечения с окружением, в которое оно будет установлено:

1. Аппаратные средства (тип и количество процессоров, объем памяти, характеристики сети / сетевых адаптеров и т. д.)

2. Программные средства (ОС, драйвера и библиотеки, стороннее ПО, влияющее на работу приложения и т. д.)

Основной упор здесь делается на тестирование с целью определения оптимальной конфигурации оборудования, удовлетворяющего требуемым характеристикам качества (эффективность, портативность, удобство сопровождения, надежность).

На следующем (клиентском) уровне программное обеспечение тестируется с позиции его конечного пользователя и конфигурации его рабочей станции. На этом этапе будут протестированы следующие характеристики: удобство использования, функциональность. Для этого необходимо будет провести ряд тестов с различными конфигурациями рабочих станций.

Тип, версия и битность операционной системы (подобный вид тестирования называется кросс-платформенное тестирование).

- 2 Тип и версия Web браузера, в случае если тестируется Web приложение (подобный вид тестирования называется кросс-браузерное тестирование).

- 3 Тип и модель видео адаптера (при тестировании игр это очень важно).

- 4 Работа приложения при различных разрешениях экрана.

- 5 Версии драйверов, библиотек и т. д. (для JAVA приложений версия JAVA машины очень важна, тоже можно сказать и для .NET приложений касательно версии .NET библиотеки) и т. д.

Порядок проведения конфигурационного тестирования. Перед началом проведения конфигурационного тестирования рекомендуется: создавать матрицу покрытия (матрица покрытия – это таблица, в которую заносят все возможные конфигурации); проводить приоритезацию

конфигураций (на практике, скорее всего, все желаемые конфигурации проверить не получится); шаг за шагом, в соответствии с расставленными приоритетами, проверяют каждую конфигурацию.

Уже на начальном этапе становится очевидно, что чем больше требований к работе приложения при различных конфигурациях рабочих станций, тем больше тестов необходимо будет провести. В связи с этим рекомендуется автоматизировать этот процесс. Конечно же, автоматизированное тестирование не является панацеей, но в данном случае оно окажется очень эффективным помощником.

Вопросы и задания для самоконтроля

- 1 Как можно измерить сложность программы?
- 2 Назовите группу видов тестирования в зависимости от цели.
- 3 На чем базируются функциональные тесты?
- 4 На каких уровнях тестирования могут быть представлены функциональные тесты?
- 5 Назовите виды функциональных тестов.
- 6 В каких аспектах может проводиться тестирование функциональности?
- 7 Каковы плюсы и минусы тестирования функциональности?
- 8 Что такое тестирование безопасности? Каковы его принципы?
- 9 Каковы критерии определения целостности?
- 10 Что такое тестирование взаимодействия?
- 11 Какой из видов тестирования функциональности чаще всего автоматизируется?
- 12 Какие существуют виды тестирования производительности? Дайте их характеристику.
- 13 Какова структура систематизации тестирования производительности? Изобразите схематично.
- 14 Какова задача тестирования производительности? Как его еще называют?
- 15 Какова задача стрессового тестирования?
- 16 Какова задача объемного тестирования?
- 17 Какова задача тестирования стабильности?
- 18 Какова задача дымового тестирования?
- 19 Перечислите типы регрессионного тестирования.
- 20 Какова задача санитарного тестирования?
- 21 В чем отличие санитарного тестирования от дымового? Являются ли они синонимами?

- 22 Аналогом какого тестирования является тестирование сборки?
- 23 Что такое тестирование установки и зачем оно необходимо?
- 24 Что такое тестирование удобства пользования? Приведите его критерии. Можно ли его измерить?
- 25 Идентично ли тестирование пользовательского интерфейса тестированию удобства пользования?
- 26 Что такое тестирование на отказ и восстановление?
- 27 Опишите сценарий (методику) тестирования на отказ.
- 28 Каковы цели конфигурационного тестирования?
- 29 Назовите уровни проведения тестирования. На что делается в них акцент? Изобразите схематично.
- 30 Какова схема проведения конфигурационного тестирования?

Литература

- 1 Бейзер, Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем / Б. Бейзер. – СПб.: Питер, 2004. – 292 с.
- 2 Благодатских, В. А. Стандартизация разработки программных средств : учебное пособие / В. А. Благодатских, В. А. Волнин, К. Ф. Посакалов; под ред. О. С. Разумова. – М.: Финансы и статистика, 2005. – 284 с.
- 3 Крылова, Г. Д. Основы стандартизации, сертификации, метрологии : учебник для вузов / Г. Д. Крылова. – 2-е изд.. – М.: ЮНИ-ТИ-ДАНА, 2001. – 356 с.
- 4 Майерс, Г. Надежность программного обеспечения / Г. Майерс. – М.: Мир, 1980. – 274 с.
- 5 Осипенко, Н. Б. Основы стандартизации и сертификации программного обеспечения: тексты лекций для студентов математических специальностей / Н. Б. Осипенко. – Гомель: ГГУ им. Ф. Скорины, 2007. – 137 с.
- 6 Тейер, Т. Надежность программного обеспечения / Т. Тейер, М. Липов, Э. Нельсон; пер. с англ. – М.: Мир, 1981. – 342 с.

Производственно-практическое издание

Осипенко Наталья Борисовна,
Осипенко Александр Николаевич

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Практическое руководство

для студентов специальности I–40 01 01
«Программное обеспечение информационных технологий»

Редактор *В. И. Шкредова*
Корректор *В. В. Калугина*

Подписано в печать 15.10.2015. Формат 60х84 1/16.

Бумага офсетная. Ризография. Усл. печ. л. 2,8.

Уч.-изд. л. 3,1. Тираж 25 экз. Заказ 606.

Издатель и полиграфическое исполнение:
учреждение образования

«Гомельский государственный университет имени Франциска Скорины».

Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/87 от 18.11.2013.

Специальное разрешение (лицензия) № 02330 / 450 от 18.12.2013.

Ул. Советская, 104, 246019, Гомель.

Н. Б. ОСИПЕНКО, А. Н. ОСИПЕНКО

**ТЕСТИРОВАНИЕ ПРОГРАММНОГО
ОБЕСПЕЧЕНИЯ**

Гомель
2015

РЕПОЗИТОРИЙ ГГУ ИМЕНИ Ф. СКОРИНЫ