2. *Орлов С.А.* Технологии разработки программного обеспечения. Разработка сложных программных систем: учебник. – СПб.: Питер, 2002. – 464 с.

3. *Черноножкин С.К.* Методы и инструменты метрической поддержки разработки качественных программ: автореферат, – Новосибирск, 1998.

4. *Богданов Д.В.* Стандартизация жизненного цикла программных средств. – СПб., 2000. – 210 с.

Using object-oriented metrics for software analysis

OlgaAlexeevnaSamoylikova, masterdegree
SiberianStateTechnologicalUniversity

Galina Mihaylovna Rudakova, PhD, professor
Siberian State Technological University,
Chair of Information Technologies.

In this paper discusses the use of object-oriented metricsfor analysis and quantification software.Quantitative evaluation complexity of the information system is made using metrics. Metrics are usually subdivided into seven classes. The most detailed description of the class of object-oriented metrics. Particular attention is paid to the set of metrics Chidamber and Kemerer.

Keywords: metrics, information system, project, class, method, encapsulation, inheritance, instance variables. set of metrics Chidamber and Kemerer.

УДК 004.054

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ – КАК УЧЕБНАЯ ДИСЦИПЛИНА

***Татьяна Ивановна Вишневская**, к.ф.-м.н., доцент кафедры ПО ЭВМ и ИТ*
Тел.: 8 499 263 6094, e-mail: tv_moscow@mail.ru
Московский государственный технический университет им. Н.Э. Баумана
http:// bmstu.ru

В статье представлены необходимые аспекты изучения стратегий тестирования и технологий автоматизированного тестирования в условиях производства программного обеспечения. Предлагаемые подходы и методики реализованы в учебном курсе «Технология программирования» в МГТУ им. Н.Э. Баумана.

Ключевые слова: тестирование программного обеспечения, модульное тестирование, интеграционное тестирование, системное тестирование, автоматизированное тестирование.

Т.И. Вишневская

С каждым годом существенно возрастает уровень сложности и важности программного обеспечения (ПО), при этом тестирование ПО становиться неотъемлемой и значительной частью процесса его создания. В настоящее время появилось большое количество учебных курсов подготовки инженеров по тестированию ПО. Эти курсы отличаются полнотой изучения вопроса, уровнем подготовки аудитории для которой они предназначены и целями, которые стояли перед авторами программ для данных курсов. Цель данной статьи сформулировать необходимые аспекты изучения стратегий тестирования и технологий автоматизации тестирования в условиях производства программного обеспечения

для студентов, обучающихся по специальности «Программная инженерия».

В основе выбора тем учебного материала по тестированию взят подход к тестированию ПО: тестировать как можно раньше, часто и в полном объёме, используя современные информационные технологии программной инженерии. Процесс тестирования переплетается с процессом разработки программ и тестировать необходимо на каждом этапе создания ПО. Раннее тестирование позволяет аналитикам и проектировщикам чётко формулировать требования к проекту и делает эти требования «поддающимися» тестированию. Исходные данные для тестирования - это техническое задание, спецификации, визуальные модели поведения программной системы и сама разрабатываемая программа, а также ее документация. Цель проектирования тестов - систематическое обнаружение различных классов ошибок при минимальных затратах времени и стоимости.

Современное ПО очень разнообразно и каждое требует своего подхода к организации тестирования. Например, структурное тестирование, объектно-ориентированное тестирование, тестирование Web-приложений и распределённых систем. Большие объёмы тестирования приводят к необходимости автоматизации тестирования.

Существуют два подхода к тестированию: по принципу «белого ящика»; по принципу «чёрного ящика» [1]. Стратегия тестирования по принципу «белого ящика» - это стратегия тестирования, управляемая логикой программы. Согласно этой стратегии подбирают такие тесты, которые обеспечат выполнение максимально возможного количества маршрутов, логических ветвлений и циклов тестируемого ПО. Стратегия тестирования, называемая стратегией «чёрного ящика» - это функциональное тестирование. Абстрагируясь от логики программы, проверяют только входные и выходные данные. Функциональное тестирование имеет целью выяснение обстоятельств, в которых поведение программы не соответствует заявленным спецификациям.

Тестирование объектно-ориентированных программ

Основные методы тестирования для объектно–ориентированных программ − это модульное тестирование (выполняет тестировщик классов); интеграционное тестирование (выполняет тестировщик целостности); системное тестирование (выполняет системный тестировщик) [3]. Модульное и интеграционное тестирования – это особые формы внутреннего базового тестирования, в которых применяют средства автоматизации.

Модульное тестирование – это тестирование классов (по принципу «белого ящика») путём разработки тестовых драйверов для проверки функциональности, входящих в класс методов. При тестировании классов тестовый драйвер создаёт один или большее число экземпляров тестируемого класса и осуществляет прогон тестовых случаев. В предлагаемом учебном курсе рассмотрены особенности тестирования классов, связанные с их свойствами: инкапсуляция, полиморфизм и наследование.

Инкапсуляция. Информацию о состоянии класса можно получить только с помощью встроенных в него методов, которые возвращают значения свойств класса. В данном случае нельзя тестировать метод изолированно, а необходимо рассматривать его как часть класса. Например, представим иерархию классов, в которой метод определён для суперкласса и наследуется несколькими подклассами. Каждый подкласс использует этот метод в контексте его приватных свойств и методов. Этот контекст меняется, поэтому данный метод надо тестировать в контексте каждого подкласса.

Полиморфизм. При вызове полиморфного метода трудно определить, какая реализация будет проверяться. Поэтому следует рассмотреть вызов метода, как в базовом, так и в производных классах.

Наследование. Новые тесты надо формировать только для тех методов, которые были переопределены в дочерних классах и не покрываются тестами для методов родительского класса.

Возможны несколько способов реализации тестового драйвера [2]:

Тестовый драйвер реализуется в виде отдельного класса. Таким способом можно тестировать часть класса с видимостью public .

Тестовый драйвер реализуется в виде класса, наследуемого от тестируемого. Такому тестовому драйверу будут доступна и часть класса с видимостью protected.

Тестовый драйвер реализуется внутри тестируемого класса (в класс добавляются новые методы). Такой драйвер имеет доступ ко всей реализации класса, включая члены с видимостью private.

Пример оформления результатов модульного тестирования приведён в табл. 1.

Таблица 1

Результат модульного тестирования

Номер теста	W-1
Название теста	DeleteTest
Тестируемый метод	NodeController.Delete
Описание теста	Проверка удаления сервера с индексом 0 из списка региональных серверов
Степень важности ошибки	Фатальная
Ожидаемый результат	Сервер с индексом 0 удаляется из списка региональных серверов и пользователь перенаправляется на домашнюю страницу
Результат теста	Тест пройден

Автоматическое тестирование предполагает создание тестового драйвера, где описана методика выполнения тестов, задающая порядок следования тестов и для каждого теста список значений параметров, который подается на вход и список результатов, ожидаемых на выходе. Контроль соответствия результатов ожидаемым (тестам) выполняется автоматически (сравнение содержимого двух файлов), и выдается сообщение «Да/Нет» для каждого теста. В настоящее время имеются различные средства для автоматизации модульного тестирования. Например, среда тестирования ***NUnit*** [5] и интегрированные средства ***Visual Studio Test Explorer*** [7] создания и выполнения модульных тестов непосредственно в среде разработчика ***Microsoft Visual Studio 2012***. Использование конкретного ПО для тестирования зависит от возможности его использования в учебных классах.

Интеграционное тестирование – тестирование по принципу «белого ящика» правильности взаимодействия классов, входящих в различные модули. Тестировщик целостности должен быть специалистом, как в области разработки программных продуктов, так и в области тестирования. Взаимодействие объектов представляет собой запрос одного объекта на выполнение другим объектом одной из операций получателя и всех видов обработки, необходимых для завершения этого запроса.

Возможны следующие способы взаимодействия классов:
-общедоступная операция имеет параметры объектного типа;
-общедоступная операция возвращает значения объектного типа;
-метод одного класса создаёт экземпляр другого класса как часть своей реализации;
-метод одного класса ссылается на глобальный экземпляр другого класса.

Выделить набор методов, посредством которых осуществляется взаимодействие классов необходимо, используя диаграмму классов, взаимодействия и кооперации, разработанные на этапе проектирования ПО.

Наиболее часто используют два направления интеграции объектно-ориентированных систем: тестирование, основанное на потоках, и тестирование, основанное на использовании [1]. Для первого направления объектом интеграции является набор классов, обслуживающий единичный ввод данных в систему. Иными словами, средства обслуживания каждого потока интегрируют и тестируют отдельно. Согласно второго - вначале интегрируют и тестируют независимые классы. Далее работают с первым слоем зависимых классов (которые используют независимые классы), со вто-

рым слоем и т.д. Выбор одного из способов тестирования зависит от особенностей конкретного ПО, например, первое направление хорошо работает для тестирования СУБД, а второе - для тестирования задач математического моделирования. Достаточность количества тестов для первого направления определяется просмотром потоков данных из всех классов эквивалентности. Для второго направления реализованы методики, позволяющие писать тестовые драйверы на основе тестирования разбиений или состояний [2].

В основу методики тестирование разбиений положен тот же подход, который применялся к отдельному классу. Отличие в том, что тестовая последовательность расширяется для включения тех операций, которые вызываются с помощью сообщений для сотрудничающих классов. Количество всех разбиений (групп методов) определяется диаграммами прецедентов разработанных на этапе проектирования ПО. Для каждого метода разбиения необходимо написать тестовый драйвер.

В качестве источника информации для тестирования на основе состояний необходимо использовать диаграммы схем состояний, фиксирующих динамику поведения класса. Данный способ позволяет получить набор тестов, проверяющих поведение класса и тех классов, которые сотрудничают с ним. Проектируемые тесты должны обеспечить покрытие всех состояний. Это значить, что тестовые варианты должны инициировать переходы через все состояния объекта. Для каждого перехода необходимо написать по тестовому драйверу. Остальные состояния можно будет протестировать аналогично. Для гарантии проверки всех вариантов поведения количество тестовых вариантов может быть увеличено. Когда поведение класса определяется в сотрудничестве с несколькими классами, для отслеживания «потока поведения» используют набор диаграмм схем состояний, характеризующих смену состояний других классов.

Пример оформления результатов интеграционного тестирования приведен в табл. 2.

Таблица 2

Результат интеграционного тестирования

Номер теста	W-2
Название теста	IndexTest1
Названия взаимодействующих классов	AdminController, User
Описание теста	Проверяется, что пользователю, который не является администратором, отказывается в доступе к домашней странице администратора
Степень важности ошибки	Фатальная
Ожидаемый результат	Пользователь будет перенаправлен на страницу «Доступ запрещён»
Результат теста	Тест пройден

Автоматическое интеграционное тестирование наиболее целесообразно проводить, например, с помощью средств, интегрированных в среду разработки Microsoft Visual Studio 2012.

Системное тестирование – это тестирование по принципу «чёрного ящика» правильности функционирования системы в целом. Системный тестировщик рассматривает ПО с точки зрения пользователя. Особую роль системное тестирование приобретает для тестирования распространённых в настоящее время Web-приложений и распределённых систем.

В процессе построения набора тестов при функциональном (системном) тестировании необходимо проверить:

-полноту реализации функциональных возможностей системы;

-эффективность защиты от искажения данных и некорректных действий;

-качество пользовательского интерфейса программы;

-тестирование практичности; тестирование БД;

-тестирование надёжности и доступности;

-корректность документации.

Системное тестирование также включает оценочное тестирование:
-тестирование удобства использования;
-стрессовое тестирование; тестирование безопасности;
-тестирование производительности на разной аппаратуре;
-тестирование удобства установки и обслуживания, совместимости.

Стрессовое тестирование производят при повышенных запросах на ресурсы системы (по количеству, частоте, размеру-объёму). Например, увеличивают скорость ввода данных; формируют варианты, требующие максимум памяти или других ресурсов; генерируют более 5 прерываний в секунду. Стрессовое тестирование обнаруживает комбинации данных, которые могут вызвать нестабильность или неправильность обработки.

Тестирование безопасности проверяет фактическую реакцию защитных механизмов, встроенных в систему, на проникновение. В ходе тестирования безопасности тестировщику разрешается найти ключ входа в систему, используя несекретные данные, и выполнить атаку системы с помощью специальных утилит, анализирующих защиты.

Тестирование производительности проверяет скорость работы ПО в компьютерной системе.

Для системного тестирования ПО реализованы следующие методики формирования тестовых наборов [2]: эквивалентное разбиение для входных данных; анализ граничных значений; предположение об ошибке, использование UML диаграммы взаимодействия, UML диаграммы деятельности, а также UML диаграммы схем состояний разрабатываемого ПО. Пример таблицы результатов системного тестирования приведён в табл. 3.

Таблица 3

Тестирование граничных значений

Номер теста	**Описание теста**	**Входные данные**	**Ожидаемые результаты**	**Реальные результаты**	**Результат теста**
W-20	Добавление нового сервера	Поле «Имя сервера»: srv02 Поле «Сетевой адрес»: (не задано) Поле «Описание»: (не задано)	Сообщение: «Поле «Адрес сервера» должно быть заполнено»	Сообщение: «Поле «Адрес сервера» должно быть заполнено»	Тест пройден

Тестирование Web-приложений и распределенных систем

Особо необходимо отметить методики системного тестирования программного обеспечения, предназначенного для работы в сети Internet [4]. Существуют четыре основных отличия между локальными и распределёнными вычислениями, которые будем учитывать при тестировании Web-приложений: реактивность (время реакции системы), латентность (разница во времени доступа к удалённым и локальным объектам), возможный частичный отказ в сети и параллелизм операции с данными.

При тестировании практичности Web-приложений необходимо оценивать дружелюбие узла к пользователям. Тестер фиксирует действия пользователя и его реакцию, чтобы определить, с каким типом трудностей сталкивается пользователь и как он их преодолевает. Тестовые примеры по практичности являются чётко заданным и несложным набором задач, которому должны следовать участники. Результаты тестирования практичности, должны быть документированы.

При тестировании форм необходимо подтвердить, что каждое поле работает так, как планировалось разработчиком. При тестировании содержимого страницы проверить, что информация, предоставляемая узлом, корректна. Тесты, подтверждающие корректность содержимого страницы с точки зрения пользователя делят на две категории: подтверждение адекватного функционирования каждой компоненты и подтверждение того, что содержимое каждой из них корректное в различных браузерах.

Оценка навигации является наиболее значимой частью тестирования практичности. При тестировании навигации необходимо проверить возможность доступа к Web-узлу, корректность перехода по всем ссылкам, адекватность открываемых изображений, возможность вернуться в предыдущее состояние или на домашнюю страницу.

Тестирование баз данных важная составляющая тестирования Web-приложений, поскольку информация на Web-узлах обычно хранится в базах данных. В некоторых приложениях есть данные, вводимые пользователем, которые затем становятся частью баз данных. К ключевым проблемам, возникающим при тестировании баз данных, относят:

- целостность данных;
- достоверность данных (подходящая форма при вводе в базы данных);
- манипуляции с данными и корректность их обновления.

Тестирование баз данных выполняют на двух уровнях: административных и пользовательских функций.

Тестирование безопасности предназначено для обнаружения уязвимых мест в приложении, которые могли бы позволить несанкционированный доступ пользователя к системе.

При тестировании надёжности и доступности необходимо оценить доступность **Web-**узла в любое время по запросу пользователя. Такая оценка достигается путём тестирования приложения в пиковое время использования (в периоды маркетинговой стимуляции и циклов высокой активности).

Производительность Web-приложений оценивается по двум «меркам»: реактивность и расширяемость [4]. ***Реактивность*** **-** это способность системы соответствовать требуемым значениям времени отклика системы. ***Расширяемость*** - это способность системы соответствовать значениям времени реакции при возрастающих требованиях к функциям программного обеспечения. Расширяемость не менее важна для поддержки реактивности при увеличении числа посетителей сайта. Значения этих параметров необходимо фиксировать и включать в документацию по тестированию.

Критерии успешности тестирования

Обязательным этапом тестирования является принятие решения о готовности ПО к эксплуатации. Команда тестировщиков отвечает за проектирование тестов, за создание сценариев автоматизированного тестирования, за выполнение тестирования и за подготовку отчётной документации по тестированию приложения. Эта команда также несёт ответственность за принятие решения о готовности приложения к эксплуатации. Для принятия решения об успешности разработки приложения необходимы критерии. Создание критериев успешности – это определение условий, при выполнении которых считается, что приложение соответствует техническому заданию. Критерии успешности иногда называют ключевыми показателями производительности. Например, для оценки неполадок или ошибок при разработке приложения в ***Microsoft .NET Framework*** рекомендуется использовать следующие параметры [4]: повторяемость (количество ошибок в единицу времени), обнаруживаемость (вероятность обнаружить ошибку), серьёзность (степень влияния неполадки на приложение, целое число в диапазоне от 1 до 10). Важность ошибки обычно вычисляется по следующей формуле:

$$Imp = (R + V) * S,$$

где Imp – важность ошибки,
R – повторяемость, V – обнаруживаемость,
S – серьезность.

Программные средства автоматического системного тестирования

К программным средствам, позволяющим выполнить автоматическое нагрузочное тестирование и тестирование производительности Web-приложений относятся, например, ***Microsoft Application Test Center***, входящий в пакет в ***MS Visual Studio 2012*** и

программа ***WAPT 4.0*** [6]. Эти программы эмулируют действия реальных пользователей действиями виртуальных пользователей. Программы позволяют проанализировать характеристики работы приложения, выявить узкие места при различных нагрузках. Действия, которые требуется выполнить приложению, описывают в сценарии тестирования. К программным средствам, позволяющим выполнить автоматическое тестирование безопасности Web-приложений, относятся, например, ***XSpider 7.8*** [6]. ***XSpider 7.***8 – это программа для анализа защищённости не только веб-приложений, но и веб-серверов.

Примером среды тестирования приложений на всех этапах его создания служат ***MS Visual Studio Test Professional 2012*** [7]. В этой среде тестировщики принимают полноценное участие в разработке приложения на всех его этапах от проекта до интерфейса, используя настраиваемое средство тестирования, фиксирующее подробности выполненных действий на каждом этапе. Имеется возможность приостановить тестирование, чтобы занести ошибку, или отправить детальный отчёт о дефекте напрямую разработчикам, предоставив все технические данные, необходимые для воспроизведения и устранения ошибки. Записывать любые тестовые действия для последующего воспроизведения, ускоряя последующее тестирование и упрощая автоматизацию.

В данной статье предложены основные стратегии тестирования ПО, конкретные методики разработки тестов и названы технологии автоматизации тестирования необходимые для изучения студентами, обучающимися по специальности «Программная инженерия».

Литература

1. Технология программирования. Основы современного тестирования программного обеспечения, разработанного на C#: учеб. пособие / под общ. ред. В.П. Котлярова. – СПб: СПбГПУ, 2004. – 168 с.

2. *Вишневская Т.И., Романова Т.Н.* Технология программирования: методические указания к лабораторному практикуму. Ч. 1. – М: МГТУ им. Н.Э. Баумана, 2007. –58 с.

3. *Макгрегор Дж., Сайкс Д.* Тестирование объектно-ориентированного программного обеспечения: практическое пособие разработчикам, менеджерам проектов, программистам. – М: DiaSoft, 2002.–432 с.

4. *Вишневская Т.И., Романова Т.Н.* Технология программирования: мет. указания к лабораторному практикуму. Ч. 2. – М: МГТУ им. Н.Э. Баумана, 2010. – 49 с.

5. *C. Poole, J. Terrell, S. Busoli.* NUnit 2012. [Электронный ресурс]. URL: http://www.nunit.org

6. XSpider 7.8 // PositiveTechnologies.2002-2012. [Электронный ресурс]. URL: http://www.ptsecurity.ru/xs7

7. MS Visual Studio Test Professional 2012 // Microsoft 2013. [Электронный ресурс]. URL: http://www.microsoft.com/visualstudio/rus/

Software Testing – Educational Program

Tatiana Ivanovna Vishnevskaya, candidate of physical and mathematical sciences, associate professor

In article necessary aspects of studying the testing strategy and technologies of automated testing in production of software are presented. Offered approaches and techniques are realized in a training course «Technology of programming» in Bauman Moscow State Technical University.

Keywords: software testing, unit testing, integration testing, the system testing, the automated testing.