Уважаемый пользователь!
Обращаем ваше внимание, что система Антиплагиат отвечает на вопрос, является ли тот или иной фрагмент текста заимствованным или нет. Ответ на вопрос, является ли заимствованный фрагмент именно плагиатом, а не законной цитатой, система оставляет на ваше усмотрение. Также важно отметить, что система находит источник заимствования, но не определяет, является ли он первоисточником.

Уважаемый пользователь!
Появление этого сообщения говорит о том, что нужно внимательнее отнестись к оценке данного документа. Документ содержит признаки, типичные для искусственного завышения процента оригинальности за счет особенностей форматов документов. Что делать: в первую очередь сравнить текст, содержащийся в отчете и в документе, отправленном на проверку. Если, например, в отчете есть текст, не видимый в исходном документе, или слова «склеены» или в слова вставлены посторонние буквы, это означает, что систему и вас пытались обмануть. В то же время, появление данного знака НЕ ОБЯЗАТЕЛЬНО свидетельствует от том, что попытка обмана была. Возможно, текст содержит слишком много иностранных или очень длинных или не найденных в словаре слов. Это часто встречается в работах, где используется много терминов (химия, юриспруденция и т.п.). В заголовке отчета дана информация, по какому критерию показан знак. НЕЛЬЗЯ ОРИЕНТИРОВАТЬСЯ ТОЛЬКО НА ПРОЦЕНТЫ И ПОЯВЛЕНИЕ ДАННОГО ЗНАКА, необходимо открывать отчет и внимательно просматривать его!

Информация о документе:

MailSubject:	MD_Gaan.pdf-21.06.2018###DB083369729095804725 82B30051BEA9
Название исходного файла:	MD_Gaan.pdf
Имя документа:	MD_Gaan.pdf
Дата проверки:	21.06.2018 17:56
Модули поиска:	Интернет (Антиплагиат), Диссертации и авторефераты РГБ, Модуль поиска ЭБС БиблиоРоссика, Университетская библиотека онлайн, Модуль поиска ЭБС "Лань", Коллекция юридических документов, Цитирования, ТУСУР

Текстовые статистики:

Индекс читаемости:	сложный
Неизвестные слова:	в пределах нормы
Макс. длина слова:	в пределах нормы
Большие слова:	в пределах нормы
Корректность файла:	**под сомнением, проверьте на соответствие текст видимый при просмотре документа и текст извлеченный из документа системой Антиплагиат**

Источник	Ссылка на источник	Коллекция/модуль поиска	Доля в отчёте	Доля в тексте
[1] Гребенюк В. М. Оценк...	http://naukovedenie.ru/PDF/13tvn113.pdf	Интернет (Антиплагиат)	5,01%	5,01%
[2] Дефекты. Причины, оп...	http://megalektsii.ru/s11941t7.html	Интернет (Антиплагиат)	2,97%	2,97%
[3] Разработка методики ...	http://dropdoc.ru/doc/324006/razrabotka-metodiki-testirovani...	Интернет (Антиплагиат)	2,84%	2,84%
[4] Непрерывная интеграц...	http://ru.wikipedia.org/wiki/%d0%9d%d0%b5%d0%bf%d1%80%d0%b5%...	Интернет (Антиплагиат)	0,76%	2,02%
[5] Разработка методики ...	http://dropdoc.ru/doc/324006/razrabotka-metodiki-testirovani...	Интернет (Антиплагиат)	0,28%	1,99%
[6] Автоматизация тестир...	http://www.dslib.net/mat-obespechenie/avtomatizacija-testiro...	Интернет (Антиплагиат)	1,92%	1,92%
[7] XUnit	http://ru.wikipedia.org/wiki/XUnit	Интернет (Антиплагиат)	1,86%	1,86%

[8] Гриппа, Генри Леонид...	http://dlib.rsl.ru/rsl01003000000/rsl01003010000/rsl01003010...	Диссертации и авторефераты РГБ	0,15%	1,79%
[9] Кафедра Вычислительн...	http://lib.knigi-x.ru/23informatika/21527-1-kafedra-vichisli...	Интернет (Антиплагиат)	0,66%	1,75%
[10] Макет ЧАСТЬ 2 2017.р...	http://docs.gsu.by/DocLib6/Conference/2017/%D0%9C%D0%B0%D0%B...	Интернет (Антиплагиат)	1,48%	1,48%
[11] Автоматизированные с...	http://helpiks.org/5-5782.html	Интернет (Антиплагиат)	0%	1,28%
[12] Технологии программи...	http://www.bibliorossica.com/book.html?&currBookId=12110	Модуль поиска ЭБС БиблиоРоссика	0%	0,9%
[13] NUnit	http://en.wikipedia.org/wiki/NUnit	Интернет (Антиплагиат)	0,89%	0,89%
[14] Лабораторная работа ...	http://rerefat.ru/docs/5/index-237178.html	Интернет (Антиплагиат)	0,68%	0,83%
[15] Разработка методики ...	http://dropdoc.ru/doc/324006/razrabotka-metodiki-testirovani...	Интернет (Антиплагиат)	0,8%	0,8%
[16] 233311	http://biblioclub.ru/index.php?page=book_red&id=233311	Университетская библиотека онлайн	0%	0,78%
[17] СИСТЕМЫ УПРАВЛЕНИЯ ...	http://sibac.info/15796	Интернет (Антиплагиат)	0,23%	0,58%
[18] Информационное общес...	http://www.kubsau.ru/upload/iblock/024/024ba6fec89c6831c5088...	Интернет (Антиплагиат)	0,28%	0,58%
[19] 5.8.2Библиотеки моду...	http://www.studfiles.ru/preview/5642268/page:38/	Интернет (Антиплагиат)	0,34%	0,5%
[20] Епифанов, Николай Ан...	http://dlib.rsl.ru/rsl01002000000/rsl01002620000/rsl01002620...	Диссертации и авторефераты РГБ	0%	0,45%
[21] Системы контроля вер...	http://studopedia.org/14-87892.html	Интернет (Антиплагиат)	0,38%	0,38%
[22] Бурыкин, Андрей Алек...	http://dlib.rsl.ru/rsl01004000000/rsl01004956000/rsl01004956...	Диссертации и авторефераты РГБ	0,01%	0,35%
[23] Прудников, Вячеслав ...	http://dlib.rsl.ru/rsl01006000000/rsl01006726000/rsl01006726...	Диссертации и авторефераты РГБ	0,03%	0,35%
[24] Инструментальные сре...	http://www.bibliorossica.com/book.html?&currBookId=19487	Модуль поиска ЭБС БиблиоРоссика	0%	0,3%
[25] Лавров, Владислав Ва...	http://dlib.rsl.ru/rsl01006000000/rsl01006769000/rsl01006769...	Диссертации и авторефераты РГБ	0%	0,28%
[26] Шамшев, Алексей Бори...	http://dlib.rsl.ru/rsl01004000000/rsl01004886000/rsl01004886...	Диссертации и авторефераты РГБ	0,03%	0,25%
[27] 64833	http://e.lanbook.com/books/element.php?pl1_id=64833	Модуль поиска ЭБС "Лань"	0,03%	0,23%
[28] Сборник ПИС-2016.pdf	https://sibsutis.ru/upload/5f9/%D0%A1%D0%B1%D0%BE%D1%80%D0%B...	Интернет (Антиплагиат)	0,1%	0,22%
[29] Геймификация процесс...	http://kpfu.ru/portal/docs/F100934898/Gejmifikaciya.processa...	Интернет (Антиплагиат)	0,14%	0,14%
[30] ДОКУМЕНТ БЕЗ НАЗВАНИ...	http://online.lexpro.ru/document/25250421	Коллекция юридических документов	0,12%	0,12%
[31] ОБ УТВЕРЖДЕНИИ ПЕРЕЧ...	http://online.lexpro.ru/document/2008464	Коллекция юридических документов	0%	0,12%
[32] 273667	http://biblioclub.ru/index.php?page=book_red&id=273667	Университетская библиотека онлайн	0%	0,11%
[33] Долбня, Николай Алек...	http://dlib.rsl.ru/rsl01006000000/rsl01006523000/rsl01006523...	Диссертации и авторефераты РГБ	0%	0,09%
[34] 58754	http://e.lanbook.com/books/element.php?pl1_id=58754	Модуль поиска ЭБС "Лань"	0%	0,09%
[35] Тестирование безопас...	http://poisk-ru.ru/s11582t2.html	Интернет (Антиплагиат)	0,07%	0,07%
[36] Источник 36		Цитирования	0,07%	0,07%

[37] ОБ УТВЕРЖДЕНИИ ГОСУД...	http://online.lexpro.ru/document/23778716	Коллекция юридических документов	0,02%	0,06%
[38] Карышева.doc		ТУСУР	0,01%	0,06%
[39] О ФЕДЕРАЛЬНОЙ ЦЕЛЕВО...	http://online.lexpro.ru/document/393994	Коллекция юридических документов	0%	0,06%
[40] О ФЕДЕРАЛЬНОЙ ЦЕЛЕВО...	http://online.lexpro.ru/document/393995	Коллекция юридических документов	0%	0,06%
[41] Киселев, Алексей Вик...	http://dlib.rsl.ru/rsl01006000000/rsl01006755000/rsl01006755...	Диссертации и авторефераты РГБ	0,01%	0,04%
[42] Фанасков, Виталий Се...	http://dlib.rsl.ru/rsl01006000000/rsl01006745000/rsl01006745...	Диссертации и авторефераты РГБ	0%	0,04%
[43] Тютин, Борис Викторо...	http://dlib.rsl.ru/rsl01007000000/rsl01007514000/rsl01007514...	Диссертации и авторефераты РГБ	0%	0,04%

Оригинальные блоки: 77,82%

Заимствованные блоки: 21,97%

Заимствование из "белых" источников: 0,21%

Итоговая оценка оригинальности: **78,03%**

Министерство образования и науки
Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (
ТУСУР)
Кафедра [38]
компьютерных систем в управлении и проектировании (КСУП)
Гаан Сергей Евгеньевич
РАЗРАБОТКА АВТОМАТИЗИРОВАННЫХ ТЕСТОВ ДЛЯ
ПРОГРАММНОГО ПРОДУКТА “ALPHA.ALARMS”
Направление 09.04.01 «Информатика и вычислительная техника»
Магистерская программа 09.04.01 «Информационное обеспечение аппаратнопрограммных комплексов»
Диссертация
на соискание степени
магистра
Научный руководитель:
Доктор технических наук,
профессор.
_____ Ю. А. Шурыгин
Консультант:
Руководитель отдела обеспечения
качества «Атомик Софт»
_____ В.Я. Гуревич
Томск 2018
2
Оглавление

3

4

Введение

Актуальность [8] работы

Появление первых компьютеров повлекло за собой и появление их непременных спутников – компьютерных программ, или программных продуктов. Программирование практически невозможно без ошибок. И если на ранних этапах развития процесс отладки программ, в силу их достаточной примитивности, не представлял большой проблемы, то в настоящее время картина происходящего изменилась.

Как известно, в процессе разработки программных продуктов программы подвергаются изменениям. Как бы хорошо они ни были написаны первоначально, в них вносятся изменения, обусловленные необходимостью исправления существующих ошибок, выявленных в процессе их выполнения, или же желанием внести в программу дополнительные изменения. Расширение областей применения программ приводит к появлению новых функциональных требований, не учтенных изначально.

Все это привело к тому, что контроль над изменениями приобрел статус критического. [6]

Исследования показали, что на долю нахождения и устранения ошибок в программном продукте приходится почти 75% всех затрат.

Для повторной проверки корректности существующей функциональности программы

необходимо использование регрессионного тестирования – повторного тестирования части программы, зависящей от внесенных изменений. Регрессионное тестирование подходит как для проверки корректности изменений, внесенных в тестируемую систему, так и для проверки корректности добавленных к системе новых компонентов и функциональных возможностей. Кроме того, регрессионное тестирование позволяет проверить, что внесенные в код программы изменения корректны и не воздействуют неблагоприятно на остальные ее части. В случае же обнаружения в программе новых ошибок считается, что в системе появились
5
регрессионные ошибки. Ввиду того, что поведение новой версии программы должно совпадать с поведением предыдущей версии, за исключением ситуаций, обусловленных внесением изменений, соответствующих новым требованиям к системе, регрессионные системные тесты можно рассматривать как частичные требования к новым версиям системы. В большинстве случаев вместо регрессионного тестирования для проверки качества новой версии программы [6] выполняются тесты, используемые [8]на этапе системного и функционального тестирования продукта.

Регрессионное тестирование относится к необходимым методам профилактического сопровождения и применяется в ходе процесса разработки и модификации программного продукта. Однако такой род деятельности является крайне ресурсоемким и, как следствие, дорогостоящим. Это обусловлено необходимостью проводить регрессионное тестирование в случае внесения даже малейших изменений в код программы, в то время, как процесс регрессионного тестирования может включать в себя исполнение достаточно большого количества тестов на скорректированной версии программы. И несмотря на то, что усилия, требуемые для внесения небольших изменений, как правило, минимальны, они могут требовать достаточно больших усилий для проверки качества измененной программы. Одним из очевидных решений здесь является автоматизация процесса тестирования, помогающая ускорить создание продукта и улучшить его качество. [6]

Так автоматизация тестирования становится все более актуальной тенденцией в разработке программного продукта во всем мире.
Объект исследования: приложение для автоматизированного тестирования программного продукта Alpha.Alarms.
Предмет исследования: автоматизированные скрипты, покрывающие проверкой рабочие функции и задачи программного продукта Alpha.Alarms. Данные скрипты составляют основу приложения для автоматизированного тестирования вышеуказанного программного продукта.
6
Цель исследования: создание программного продукта по автоматизированному тестированию, способного за приемлемое время производить анализ качества, функций, задач тестируемого программного продукта Alpha.Alarms компании АО «Атомик Софт».
Задачи исследования:
1) изучить понятие тестирования ПО, классификацию видов тестирования, а также методологию и процесс тестирования;
2) изучить понятие непрерывной интеграции, а также проанализировать популярные CI платформы;
3) проанализировать популярные фреймворки автоматизированного тестирования;
4) провести обзор наиболее распространенных систем контроля версий;
5) изучить программный продукт Alpha.Alarms компании АО «Атомик Софт»;
6) рассмотреть уровни автоматизации тестирования и архитектуру скриптов;
7) проанализировать экономическую рентабельность и временную эффективность автоматизации тестирования выбранного программного продукта;
8) разработать приложение для автоматизированного тестирования программного продукта Alpha.Alarms.
Методологической основой работы является комплексный подход.
Для решения поставленных задач применялись следующие методы:
1) анализ литературных источников;
2) классификация тестирования;
3) моделирования поведения пользователя и работа основного функционала программного продукта Alpha.Alarms.
Практическая и научная значимость: состоит в сокращении издержек на проведение тестирования программного продукта Alpha.Alarms как по окончанию разработки, так и при выпуске обновлений.
7
С учётом высокой стоимости разработки и поддержки информационных систем, в которые также входит и стоимость тестирования, а также затрачиваемого времени, проведенная работа имеет высокую актуальность, а достигнутые результаты могут иметь существенную практическую пользу.
Границы исследования: Основной акцент в данном исследовании ставится на разработку и анализ автоматизированных тестов. Приложение

содержащие вышеупомянутые автоматизированные скрипты написано объектно-ориентированном языке программирования C# с использованием фреймворка тестирования NUnit.
Апробация результатов: подтверждается соответствующими актами о внедрении разработанного приложения для проведения автоматизированного тестирования.
Основные публикации по теме исследования. По материалам работы были подготовлены и сделаны доклады:
1) Сборник тезисов XXIV Международной научной конференции студентов, аспирантов и молодых учёных «ЛОМОНОСОВ–2017» секция «ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА И КИБЕРНЕТИКА»
2) 23-я Всероссийская научно-техническая конференция «Научная сессия ТУСУР-2018»
Структура работы: Диссертационная работа состоит из введения, трех глав, заключения и списка [23]литературы. [41]

Материал изложен на 110 страницах, включает 16 таблиц, 32 рисунков, а также одно приложение. Список использованной информации содержит 42 наименование.
Первая глава посвящена изучению теоретической базы, которая будет использоваться в дальнейшем при разработке автоматизированных тестов.
Во второй главе рассмотрена сама программа для которой планируется написание автоматизированных тестов Alpha.Alarms. Так же проведен расчет экономической и временной эффективности автоматизированных тестов в сравнении с ручным тестирование.
8
В третьей главе приведено развернутое описание получившегося проекта для автоматизированного тестирования программного продукта Alpha.Alarms.Рассмотрены различные виды запуска автоматизированных тестов. Так же была произведена оценка тестового покрытия программы автоматизированными тестами на текущий момент.
9
Глава 1. Теоретическая часть
1.1 Теоретические аспекты процесса тестирования
Данная глава посвящена решению таких задач, как: выявление теоретических основ тестирования, классификация и описание видов тестирования, анализ и описание процесса тестирования, выявление критериев корректно построенного процесса [1].
Решение данных задач необходимо для того, чтобы лучше понимать процессы тестирования, различать их виды и применять эти знания при оценке целесообразности внедрения автоматизированного тестирования в компании.
1.1.1 Определение понятия тестирования ПО

Тестирование программного обеспечения – это проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранных определенным образом. Тестирование – это одна из техник контроля качества, [29] которая включает в себя такие процессы, как: проектирование тестов, выполнение тестирования и анализ полученных результатов.
Общая схема тестирования представлена на рисунке 1.1.
Рисунок 1.1– Общая схема тестирования
На входе тестировщик получает программу, которую необходимо тестировать и требования к ней. Наблюдая за программой в определенных условиях, на выходе тестировщик получает информацию о соответствии или
10
несоответствии программы требованиям. Данная информация используется для исправления ошибок в существующем продукте, либо для изменения требований к продукту, который находится на стадии разработки.
Тест (проверка) включает в себя выбранную определенным образом

искусственно созданную ситуацию и описание наблюдений, которые нужно осуществить для проверки программы на соответствие определенным требованиям.
Тест может быть, как коротким, так и длинным, например, тест производительности, проверяющий работоспособность системы при длительной нагрузке.
Таким образом, в процессе тестирования тестировщик выполняет две основные задачи.
Первой задачей является управление рабочим процессом программы, а также создание искусственных ситуаций, в которых и происходит проверка поведения программы.
Вторая задача состоит в наблюдении за тем, как программа ведет себя в различных созданных ситуациях, и в сравнении того, что он видит с тем, что ожидается.
Если рассматривать задачи современного тестирования, то можно прийти к выводу, что они заключаются не только в обнаружении ошибок в программах, но и в выявлении причин, по которым они возникают. Такой подход к процессу тестирования позволяет разработчикам выполнять свою работу с максимальной эффективностью, устраняя обнаруженные ошибки быстро и своевременно [2].
1.1.2 Классификация видов тестирования
При тестировании программного продукта применяется огромное количество различных видов тестов. Существует несколько классификаций видов тестирования. Наиболее широкую и подробную предложил автор книги «Тестирование Дот Ком» Роман Савин. Он объединил виды
11
тестирования по следующим признакам: объект, субъект тестирования, уровень, позитивность тестирования и степень автоматизации тестирования.
Классификация, представленная в данном исследовании основана на вышеупомянутой книге и также была дополнена на основании информации из таких источников, как: книга Сэма Канера, «
Тестирование программного
обеспечения»
и интернет-ресурс, посвященный тестированию, «
Про Тестинг
– Тестирование Программного Обеспечения» [3].
1.1.2.1
По объекту тестирования
1) Функциональное тестирование. Функциональное тестирование на сегодняшний день является одним из наиболее часто применяемых видов тестирования. Его задача – установить, насколько соответствует разработанное программное обеспечение (ПО) требованиям заказчика с точки зрения функционала. Иначе говоря, проведение функциональных тестов позволяет проверить способность информационной системы решать задачи пользователей.
2) Нефункциональное тестирование. Позволяет проверить соответствие свойств программного обеспечения обозначенным нефункциональным требованиям. Другими словами, нефункциональное тестирование – это тестирование всех свойств программы, не относящихся к функциональности системы. Такими свойствами могут быть предъявленные характеристики с точки зрения таких параметров как:
а) надежность (способность системы реагировать на непредвиденные ситуации);
б) производительность (способность системы работать в условиях больших нагрузок);
в) удобство (исследование удобства работы пользователя с приложением);
г) масштабируемость (возможность масштабировать приложение как вертикально, так и горизонтально);
12
д) безопасность (исследование возможности нарушения работы приложения и кражи пользовательских данных злоумышленниками);

е) портируемость (возможность перенести приложение на определенный набор платформ).

Этот список можно дополнить и рядом других качеств.

1) Тестирование пользовательского интерфейса. Это тестирование корректности отображения элементов пользовательского интерфейса на различных устройствах, правильности реагирования их на совершение пользователем различных действий. Также данный вид тестирования позволяет оценить, насколько ожидаемо ведет себя программа в целом. Кроме того, такое тестирование дает возможность выявить, насколько эффективно пользователь сможет работать с приложением и насколько внешний вид приложения соответствует утвержденным документам, созданными дизайнерами. При проведении тестирования пользовательского интерфейса основной задачей тестировщика является выявление визуальных и структурных недостатков в графическом интерфейсе приложения, проверке возможности и удобства навигации в приложении и корректность обработки приложением ввода данных с клавиатуры, мыши и других устройств ввода. Тестирование пользовательского интерфейса необходимо проводить для того, чтобы убедиться в том, что интерфейс соответствует утвержденным требованиям и стандартам, а также чтобы гарантировать возможность работы пользователя с графическим интерфейсом приложения.

2) Тестирование удобства использования. Это способ тестирования позволяет оценить степень удобства использования приложения, скорость обучения пользователей при работе с программой, а также понятность и привлекательность программного продукта для пользователей в контексте заданных условий. Данное тестирование необходимо для обеспечения максимально положительного пользовательского опыта при работе с приложением.

13

3) Тестирование защищенности. Такое тестирование позволяет выявить главные уязвимости программного обеспечения по отношению к различным атакам со стороны злоумышленников. Компьютерные системы довольно часто подвергаются кибер-атакам с целью нарушения работоспособности информационной системы либо кражи конфиденциальных данных. Тестирование безопасности дает возможность проанализировать реальную реакцию и действенность защитных механизмов, использованных в системе, при попытке проникновения в неё извне. В процессе тестирования безопасности тестировщик имитирует действия взломщика программы. При попытке тестировщиком взломать систему могут использоваться любые средства: атаки системы при помощи специальных утилит; попытки узнать логины и пароли с помощью внешних средств; DDOS атаки; целенаправленная генерация ошибок для обнаружения возможности проникновения в систему в процессе её восстановления; использование известных незакрытых уязвимостей системы.

4) Инсталляционное тестирование. Под этим термином подразумевают тестирование корректности установки (инсталляции) определенного программного продукта. Данное тестирование происходит, как правило, в искусственно созданных средах. Его цель – выявить степень готовности программного обеспечения к эксплуатации. Основные причины проведения таких тестов связаны с необходимостью проверить корректность поведения программного продукта при автоматизированном развертывании либо обновлении. Обеспечение правильной и стабильной установки программного обеспечения является важным фактором при создании программного продукта, поскольку позволяет пользователям быстрее и с меньшими усилиями начать использовать продукт, рассчитывая на одинаково корректное поведение этого продукта во всех протестированных программных средах.

5) Конфигурационное тестирование. Данное тестирование предназначено для оценки работоспособности программного обеспечения

14

при разнообразных конфигурациях системы. В зависимости от типа тестируемого программного продукта, оно может преследовать разные цели. Чаще всего это либо определение оптимальной конфигурации оборудования, обеспечивающего достаточные для работы ПО параметры

производительности, либо проверка определенной конфигурации оборудования (или платформы, включающей в себя помимо оборудования, стороннее ПО, необходимое для работы программы) на совместимость с тестируемым продуктом. Если речь идет о клиент-серверном программном обеспечении, то конфигурационное тестирование проводится отдельно для сервера и отдельно для клиента. При тестировании совместимости сервера с определенной конфигурацией чаще всего стоит задача найти оптимальную конфигурацию, поскольку важна стабильность работы и производительность сервера. В то время как при тестировании клиента, наоборот, тестировщики стремятся выявить недостатки ПО при любых конфигурациях и по возможности устранить их.

6) Тестирование надежности и восстановления после сбоев (стрессовое тестирование). Такой вид тестирования довольно часто проводится для программного обеспечения, работающего с ценными пользовательскими данными, поскольку его бесперебойная работа и скорость восстановления после сбоев крайне важны для пользователя. Тестирование на отказ и восстановление осуществляет проверку способности программы быстро и успешно восстанавливаться после отказа оборудования, перебоев сети или критических ошибок в самом программном обеспечении. Это дает возможность оценить последствия отказа и время, необходимое для последующего восстановления системы. На основе полученных в ходе тестирования данных может быть оценена надежность системы в целом, и при условии неудовлетворительных показателей могут быть приняты соответствующие меры, направленные на улучшение систем восстановления.

7) Тестирование локализации. Данное тестирование дает возможность выяснить насколько хорошо приспособлен продукт для населения

15

определенных стран и насколько он соответствует ее культурным особенностям. Чаще всего рассматриваются культурный и языковой нюансы, а именно перевод пользовательского интерфейса, сопутствующей документации и файлов на определенный язык, также тестируется правильность форматов валют, чисел, времени и телефонных номеров.

8) Нагрузочное тестирование. Такое тестирование позволяет выявить максимальное количество однотипных задач, которые программа может выполнять параллельно. Самая популярная цель нагрузочного тестирования в контексте клиент-серверных приложений – оценить максимальное количество пользователей, которые смогут одновременно пользоваться услугами приложения.

9) Тестирование стабильности. Данный вид тестирования проверяет работоспособность приложения при длительном использовании на средних нагрузках. В зависимости от типа приложения, формируются определенные требования к длительности его бесперебойной работы. Тестирование стабильности стремится выявить такие недочеты приложения, как: утечки памяти, наличие ярко выраженных скачков нагрузки и прочие факторы, способные помешать работе программного продукта в течение длительного периода времени.

10) Тестирование масштабируемости. Это вид тестирования программного обеспечения, предназначенный для проверки способности продукта к увеличению (иногда к уменьшению) масштабов определенных нефункциональных возможностей. Некоторые виды приложений должны легко масштабироваться и, при этом, оставаться работоспособными и выдерживать определенную пользовательскую нагрузку.

11) Объемное тестирование. Задача данного вида тестирования состоит в выявлении реакции приложения и в оценке возможных ухудшений в работе ПО при значительном увеличении количества данных в базе данных приложения. Как правило, в такое тестирование входит:

16

a) замер времени выполнения операций, связанных с получением или изменением данных БД при определенной интенсивности запросов;

b) выявление зависимости увеличения времени операций от объема данных в БД;

c) Определение максимального количества пользователей, которые имеют возможность одновременно работать с приложением без заметных

задержек со стороны БД [4].

1.1.2.2 Тестирование, связанное с изменениями

1) Санитарное тестирование. Это один из видов тестирования, цель которого – доказать работоспособности конкретной функции или модуля в соответствии с техническими требованиями, заявленными заказчиком. Санитарное тестирование используется, как правило, при проверке какойлибо части программы или приложения при внесении в нее определенных изменений со стороны факторов окружающей среды. Данный вид тестирования чаще всего выполняется в ручном режиме.

2) Дымовое тестирование. Данный вид тестирования представляет собой короткий цикл тестов, цель которых состоит в подтверждении факта запуска и выполнения функций устанавливаемого приложения после того, как новый или редактируемый код прошел сборку. По завершении тестирования наиболее важных сегментов приложения предоставляется объективная информация о присутствии или отсутствии дефектов в работе тестируемых сегментов. По результатам дымового тестирования принимается решение об отправке приложения на доработку или о необходимости его последующего полного тестирования.

3) Регрессионное тестирование. Это тестирование, направленное на обнаружение ошибок в уже протестированных участках. Регрессионное тестирование проверяет продукт на ошибки и сбои, которые могли появиться в результате добавления нового участка программы или исправления других ошибок. Цель данного вида тестирования – убедиться, что обновление

17

сборки или исправление ошибок не повлекло за собой возникновения новых багов [5].

1.1.2.3 По уровню тестирования

1) Модульное тестирование (Unit тесты). Оно заключается в проверке каждого отдельного модуля (самобытного элемента системы) путем запуска автоматизированных тестов в искусственной среде. При реализации таких тестов часто используют различные заглушки и драйверы для имитации работы реальной системы. Модульное автоматизированное тестирование – это возможность запустить и проверить исходный код. Создание Unit тестов для всех модулей системы позволяет очень быстро выявлять ошибки в коде, которые могут появиться в ходе разработки.

2) Интеграционное тестирование. Это тестирование отдельных модулей системы на предмет корректного взаимодействия. Основная цель интеграционного тестирования – найти дефекты и выявить некорректное поведение, связанное с ошибками в интерпретации или реализации взаимодействия между модулями.

3) Системное тестирование. Это тестирование программы в целом, позволяющее проверять соответствие программы заявленным требованиям.

4) Приемочное тестирование. Это комплексное тестирование, определяющее фактический уровень готовности системы к эксплуатации конечными пользователями. Тестирование проводится на основании набора тестовых сценариев, покрывающих основные бизнес-операции системы [6].

1.1.2.4 По исполнению кода

1) Статическое тестирование. Это выявление артефактов, появляющихся в процессе разработки программного продукта путем анализа исходных файлов, таких как документация или программный код. Такое тестирование проводится без непосредственного запуска кода, качество исходных файлов и их соответствие требованиям оцениваются вручную, либо с использованием вспомогательных инструментов. Статическое тестирование должно проводиться до динамического тестирования, таким

18

образом, ошибки, обнаруженные на этапе статического тестирования, обойдутся дешевле. С точки зрения исходного кода, статическое тестирование выражается в ревизии кода. Обычно ревизия кода отдельных файлов производится после каждого изменения этих файлов программистом. Также ревизия может проводиться как другим программистом, так и ведущим разработчиком, либо отдельным работником, занимающимся ревизией кода. Использование статического тестирования дает возможность поддерживать качество программного обеспечения на всех стадиях

разработки и уменьшает время разработки продукта.
2) Динамическое тестирование. В отличие от статического тестирования, такой вид тестирования предполагает запуск исходного кода приложения. Другими словами, динамическое тестирование содержит в себе множество других типов тестирования, которые уже были описаны выше. Динамическое тестирование позволяет выявить ошибки в поведении программы с помощью анализа результатов ее выполнения. Таким образом, большая часть существующих типов тестирования относятся к классу динамического тестирования [7].
1.1.2.5 По субъекту тестирования
1) Альфа-тестирование. Это тестирование проводится для самых ранних версий программного продукта (или аппаратного устройства). Альфатестирование почти всегда проводится самими разработчиками ПО. В процессе альфа-тестирования разработчики приложения находят и исправляют ошибки и проблемы, имеющиеся в программе. Как правило, во время Альфа-тестирования происходит имитация работы с программой штатными разработчиками, реже имеет место реальная работа как потенциальных пользователей, так и заказчиков с продуктом. Чаще всего альфа-тестирование проводится на самом раннем этапе разработки ПО, однако в отдельных случаях может быть применено для законченного или близкого к завершению продукта, например, в качестве приёмочного тестирования.
19
2) Бета-тестирование. Это вид тестирования продукта, по-прежнему находящегося в стадии разработки. При бета-тестировании этот продукт предоставляется для некоторого количества пользователей, для того чтобы его изучить и сообщить о возникающих проблемах, с которыми сталкиваются пользователи. Такое тестирование необходимо проводить для того, чтобы найти ошибки, которые разработчики могли пропустить. Как правило, бета-тестирование проводится в две фазы: закрытый бета-тест и открытое бета-тестирование. Закрытый бета-тест – это тестирование на строго ограниченном кругу избранных пользователей. Такими пользователями могут выступать знакомые разработчиков либо их коллеги, не связанные напрямую с разработкой тестируемого продукта. Открытое бета-тестирование заключается в создании и размещении в открытом доступе публичной бета-версии. В данном случае любой пользователь может выступать бета-тестером. Обратная связь от таких бета-тестеров осуществляется с помощью отзывов на сайте и встроенных в программу систем аналитики и логирования пользовательских действий. Такие системы необходимы для анализа поведения пользователей и обнаружения трудностей и ошибок, с которыми они сталкиваются [8].
1.1.2.6 По позитивности сценария
1) Позитивное тестирование. Тесты с позитивным сценарием проверяют способность программы выполнять заложенный в нее функционал. Как правило, для такого вида тестирования разрабатываются специальные тестовые сценарии, при выполнении которых, в нормальных для ПО условиях работы, не должно возникать никаких сложностей.
2) Негативное тестирование. Данный вид тестирования программного продукта происходит на сценариях, соответствующ��х некорректному поведению программы. Такие тесты проверяют корректность работы программы в экстренных ситуациях. Это позволяет удостовериться в том, что программа выдает правильные сообщения об ошибках, не повреждает пользовательские данные и ведет себя корректно при ситуациях, в которых
20
не предусмотрено штатное поведение продукта. Основная цель негативного тестирования – проверить устойчивость системы к различным воздействиям, способность правильно валидировать входные данные и обрабатывать исключительные ситуации, возникающие как в самих программных алгоритмах, так и в бизнес-логике [9].
1.1.2.7 По степени автоматизации
1) Ручное тестирование. Данное тестирование проводится без использования дополнительных программных средств, оно позволяет проверить программный продукт или сайт с помощью имитации действий

пользователя. При таком тестировании тестировщик выступает в качестве пользователя, следуя определенным сценариям, параллельно анализируя вывод программы и ее поведение в целом.
2) Автоматизированное тестирование. Такой вид тестирования позволяет значительно ускорить процесс тестирования за счет использования дополнительного программного обеспечения для автоматизации тестов. Такое дополнительное ПО позволяет контролировать и управлять выполнением тестов и сравнивать ожидаемый и фактический результаты работы программы. Более подробно автоматизированное тестирование будет рассмотрено позже [10].
21
Рисунок 1.2– Основные виды тестирования
1.1.3 Методологии тестирования
Существуют различные методологии динамического тестирования ПО. В зависимости от наличия у тестировщика доступа к исходному коду программы, выделяют следующие методы тестирования [11]:
1) метод черного ящика;
2) метод белого ящика;
3) метод серого ящика.
1.1.3.1 Метод черного ящика
Впервые термин «черный ящик» упоминается психиатром У. Р. Эшби в книге "Введение в кибернетику" в 1959 г. Он пишет, что метод черного ящика позволяет изучать поведение системы, абстрагируясь от ее внутреннего устройства.
22
В области тестирования метод черного ящика – это техника тестирования, которая основана на работе с внешними интерфейсами программного обеспечения, без знания внутреннего устройства системы. Данный метод назван “Черным ящиком”, потому что при его использовании тестируемое программное обеспечение для тестировщика выглядит как закрытый ящик, внутри которого происходят некоторые процессы, о которых тестировщику принципиально ничего не известно. Данная техника позволяет обнаружить ошибки в следующих категориях:
1) ошибки интерфейса;
2) недостающие или неправильно реализованные функции;
3) недостаточная производительность или ошибки в поведении системы;
4) некорректные структуры данных или плохая организация доступа к внешним базам данных.
Таким образом, поскольку тестировщик не имеет никакого представления о внутреннем устройстве и структуре системы, ему необходимо сконцентрироваться на том, что делает программа, а не на том, каким образом она это делает [12].
1.1.3.2 Метод белого ящика
Данный метод тестирования противоположен методу черного ящика. Он основан на анализе внутренней структуры системы.
В данном случае тестировщику известны все аспекты реализации тестируемого программного обеспечения. Этот метод позволяет протестировать не только корректность реакции программы на определенный ввод (как в случае с черным ящиком), но и правильную работу отдельных модулей и функций, основываясь на знании кода, который обрабатывает этот ввод. Знание особенностей реализации тестируемой программы – обязательное требование к тестировщику для успешного применения этой техники. Тестирование методом белого ящика позволяет углубиться во внутренне устройство ПО, за пределы его внешних интерфейсов [13].
23
1.1.3.3 Метод серого ящика
Этот метод тестирования системы предполагает комбинацию подходов Белого и Черного ящиков. Это нужно в тех случаях, когда тестировщику лишь частично известно внутреннее устройство программы. Например, предполагается наличие доступа к внутренней структуре программного обеспечения для разработки максимально эффективных тест-кейсов, в то время как само тестирование будет проводится методом черного ящика. Или

тестировщики могут во всем следовать методу черного ящика, однако для того, чтобы убедиться в корректной работе отдельных алгоритмов, могут смотреть информацию в логах или анализировать записи программы в базе данных [14].

1.1.4 Процесс тестирования

Тестирование представляет собой процесс проверки того, насколько программное обеспечение соответствует требованиям, заявленным заказчиком. Оно осуществляется в специальных искусственно создаваемых ситуациях посредством наблюдения за работой программного обеспечения. Подобные искусственно созданные ситуации называют тестовыми ситуациями или тестами.

Разработка тестов происходит на основе проверяемых требований и критерия полноты [5]

тестирования. Разработанные скрипты формируются в набор тестов и выполняются

на ПО, которое нужно протестировать. После выполнения всех тестов анализируется результат, в результате чего можно выявить ошибки в программе [15].
[5]

Процесс тестирования схематично показан на рисунке 1.3.

24

Рисунок 1.3– Процесс тестирования

1.1.4.1 Разработка тест-кейсов

Тест-кейс (с англ. Test Case – тестовый случай) – это минимальный элемент тестирования (одна проверка), содержащий в себе описание конкретных действий, условий и параметров, которые направленны на проверку какой-либо функциональности. Набор тест-кейсов называется тестовым набором (test suite).

Тест-кейсы позволяют тестировщикам провести проверку продукта без полного ознакомления с документацией. Если созданный тест-кейс удобен в поддержке, то, написанный один раз, он позволит сэкономить большое количество времени и усилий тестировщиков. Подробные тест-кейсы также способны существенно снизить вариативность выполнения тестов различными тестировщиками, что повышает качество тестирования [16].

1.1.4.2 Атрибуты тест-кейса

Тест-кейс должен включать в себя:

1) уникальный идентификатор тест-кейса. Он необходим для удобства организации и навигации по тестовым наборам;

2) название. В нём должна отражаться основная идея тест-кейса, цель данной проверки;

3) предусловия. Это список шагов, которые необходимо выполнить до начала теста. При этом, они не имеют прямого отношения к проверяемому

25

функционалу. Например, для тест-кейса «Заказ товара» предусловием может быть шаг «авторизоваться на сайте», если на данном сайте заказать товар может только авторизованный пользователь;

4) шаги. Описание последовательности действий, которая должна привести к ожидаемому результату;

5) история редактирования. Она дает возможность узнать, кем и когда был изменен тест-кейс. Это удобно, поскольку позволяет более эффективно редактировать и поддерживать тест-кейсы;

6) ожидаемый результат. Это поведение системы, предусмотренное требованиями. Один тест-кейс проверяет одну конкретную функцию, поэтому ожидаемый результат может быть только один;

7) статус кейса. Он проставляется в соответствии с тем, соответствует ли фактический результат ожидаемому. Тест-кейс может иметь один из трех статусов:

a) положительный, если фактический результат совпадает с ожидаемым результатом;

b) отрицательный, если фактический результат не совпадает с

ожидаемым результатом. Если статус кейса отрицательный, это означает, что в процессе его проведения была обнаружена какая-либо ошибка в работе тестируемого продукта;
с)

выполнение теста блокировано, если после одного из шагов продолжение теста невозможно. В этом случае так же, [35]

обнаруживается
ошибка в работе программы [17].
1.1.4.3 Требования к тест-кейсу
1) При разработке тест-кейса для определенного функционала, требования к продукту должны быть проанализированы и впоследствии разбиты на пункты. Если тест-кейсы покрывают все требования, то может быть дан положительный или отрицательный ответ о реализации данного требования в продукте.
26
2) Тест является положительным в случае, когда он может обеспечить высокую вероятность обнаружения ошибки. Показать, что в программе полностью отсутствуют ошибки невозможно, поэтому процесс тестирования должен быть нап��авлен на выявление прежде не найденных ошибок.
3) Четкие, однозначные формулировки шагов. Описание шагов для прохождения тест-кейса должно содержать всю необходимую информацию, но при этом оно не должно быть слишком детализировано. Например, если тест-кейс содержит такие шаги, как авторизация, в описании необходимо указывать логин и пароль, но не нужно указывать в каком углу экрана находится окно авторизации.
4) Отсутствие зависимостей тест-кейсов. Если тесты связаны между собой, становится проблематичным изменение, дополнение или удаление конкретного тест-кейса и появляется необходимость изменять связанные с ним тесты. Более того, взаимосвязанные тесты обладают конкретным сценарием от перехода одного теста к другому. Изменения в каком-либо из связанных тест-кейсов приведут к тому, что не все сценарии перехода от одного теста к другому будут протестированы и появится вероятность пропустить ошибку в работе тестируемого продукта.
5) Ожидаемый результат необходимо прогнозировать заранее и прописывать его в тест-кейсе. Если этого не сделать, возникает вероятность вольной трактовки результата теста тестировщиком. При заранее определенном результате тестировщику необходимо только сравнить ожидаемый результат с фактическим.
6) Необходимо уделять внимание как положительным тестам, которые проверяют правильные данные, так и тестам, которые тестируют работу программы при ошибочных, непредусмотренных данных и конфигурациях. Большое количество ошибок зачастую связано с теми действиями пользователя, которые не предусмотрены программой [18].
27
7) Кроме того,

необходимо проверять, не делает ли программа то, [5]

чего
выполнять не должна. Также нужно производить проверку на нежелательные побочные эффекты.
1.1.4.4 Выполнение тест-кейсов
Одной из особенностей процесса тестирования является необходимость проведения тестирования программы специалистом, который не является ее автором. Неприемлемо тестирование продукта программистом, создавшим его. Это правило должно быть применимо ко всем, без исключения, формам тестирования, например, как к тестированию целой системы, так и к тестированию отдельных модулей, а также внешних функций. Это нужно для более качественной проверки, так как при проведении тестирования не автором программы, он становится более точным и эффективным.
Если рассматривать суть процесса тестирования, то становится очевидным, что это процесс деструктивный. С этой его особенностью и связывают представление о тестировании как о сложной работе. Таковой тэта

работа представляется для программиста, создавшего продукт, так как проектирование, разработка и написание программы является процессом конструктивным, в отличие от тестирования, в процессе которого специалисту необходим настрой на деструктивный образ мышления. Исходя из этой особенности тестирования, приступать к качественному и точному выявлению ошибок сразу же по завершении создания программы представляется трудновыполнимым для ее автора.

Кроме того, разработчик программного продукта может при разработке тестов допустить в них те же самые ошибки, что и при разработке самой программы. Например, содержащиеся в модуле дефекты, которые являются следствием ошибок перевода (например, неверная интерпретация спецификации), с высокой долей вероятности будут присутствовать и в тестах, так как оба процесса будут выполняться одним и тем же специалистом.

28

Однако не следует делать однозначный вывод, что тестирование программы специалистом, ее создавшим, невозможно. Большое количество программистов справляются с этой задачей вполне успешно. Но, исходя из вышесказанного, можно прийти к заключению, что выполнение тестирования другим специалистом, не являющимся автором тестируемого продукта, гораздо эффективнее и плодотворнее. Поэтому создаются группы тестировщиков для проведения процесса тестирования с наиболее оптимальными результатами.

Принятие решения о сроках остановки тестирования является одной из наиболее сложных проблем при проведении тестирования. Это происходит, потому что невозможно провести исчерпывающее тестирование или, другими словами, процесс испытания всех входных значений. Так, при процессе тестирования техническая сторона проводимых действий входит в противоречие с экономической выгодой, так как возникает вопрос выбора оптимального конечного количества тестов, которое дает максимально возможный результат при определенных затратах.

Принимая решение о количестве времени и проводимых тестов с программой, всегда необходимо принимать во внимание уровень риска, как технического, так и бизнес риска, для отдельно взятого продукта и для всего проекта. Более того, нужно учитывать тот факт, что проект обычно имеет ограничения по времени и бюджету [19].

1.1.4.5 Анализ результатов тестирования

Несмотря на существование различных видов тестирования, процессы в них достаточно схожи. Разработкой и анализом тестов может заниматься только тестировщик. За выполнение тест-кейсов так же отвечает тестировщик, их выполнение может производиться как вручную, так и в автоматизированном режиме.

По результатам выполнения каждого теста, ему присваивается статус (положительный, отрицательный, блокирован). Если тест получает отрицательный статус, то в зависимости от методологии тестирования

29

тестировщик может проводить дополнительную работу для выявления конкретной ошибки, которая была причиной некорректного поведения программы.

При использовании методологии черного ящика, анализ результатов теста сводится к выявлению общих закономерностей, ведущих к появлению ошибки. Однако, когда используется метод белого или серого ящика, тестировщик может проводить гораздо более глубокий анализ причин возникновения ошибки. В зависимости от доступных ему данных (база данных, исходный код программы, логи и т.п.), он способен с некоторой точностью определить источник некорректного поведения программы. После того, как максимально точно выявлена причина нежелательного поведения, тестировщик должен описать её вместе со способом воспроизведения ошибки для дальнейшей передачи этой информации разработчикам. Когда источник ошибки точно определен и хорошо описан, разработчики могут гораздо быстрее устранить обнаруженную ошибку [20].

1.2 Описание и анализ дефектов

Ошибки в программном обеспечении – это

все возможные
[5]несоответствия между демонстрируемыми характеристиками программного продукта и сформулированными или подразумеваемыми требованиями и ожиданиями пользователей.
В англоязычной литературе используется несколько терминов, часто одинаково переводящихся как "ошибка" на русский язык:
1) defect — самое общее нарушение каких-либо требований или ожиданий, [2] которое не обязательно проявляется вовне (пример: нарушение [26]стандартов кодирования, недостаточная гибкость системы и пр.);
2) failure — наблюдаемое нарушение требований, проявляющееся при каком-то реальном сценарии работы ПО. Это также можно назвать проявлением ошибки;
3) fault — ошибка в коде программы, вызывающая нарушения требований при работе. [2]

Другими словами, это то место кода, которое
30
нуждается в исправлении. Несмотря на частоту использования этого понятия, его смысл размыт, поскольку для устранения нарушения в некоторых случаях нужно исправить программу в нескольких местах;
4) error — используется в двух смыслах. Во-первых, это ошибочное представление программиста о программе, которое привело к появлению ошибок и неточностей в коде программного продукта. Во-вторых,

это
некорректные значения данных (выходных или внутренних), которые возникают при ошибках в работе программы [21].
Эти понятия связаны с основными источниками ошибок. Поскольку при разработке [2]

программного продукта программист совершает одинаковый алго��итм действий (а именно: он должен понять задачу, найти решение для нее и перевести его в код программы), то можно назвать три основных источника ошибок:
1) неправильное понимание задач. Разработчики ПО не всегда корректно понимают требования в задаче. С другой стороны, понимание задач может отсутствовать и у самих заказчиков и пользователей: достаточно часто они заявляют в требованиях не то, что ожидают от разработчиков в результате. Для предотвращения неправильного понимания задач при разработке программного продукта необходимо проводить анализ предметной области задачи;
2) неправильное решение задач. В некоторых случаях разработчики могут выбрать неправильный подход к решению поставленной задачи. Так, выбранные способы

решения могут обеспечивать лишь некоторые из требуемых свойств, то есть, они могут хорошо подходить для данной задачи в теории, но плохо – на практике, в конкретных обстоятельствах, в которых должно работать программное обеспечение. [2]

Для того, чтобы избежать
выбора неправильных способов решений задач, необходимо сопоставлять альтернативные решения и тщательно анализировать их на предмет соответствия всем требованиям. Также принять верное решение поможет постоянное поддержание связи с заказчиками и пользователями,
31

предоставление им необходимой информации о выбранных решениях, демонстрация прототипов, анализ пригодности выбираемых решений для работы в том контексте, в котором они будут использоваться;
3) неправильный перенос решений в код. Имея правильное решение [2]

корректно понятой задачи, разработчик способен по ряду причин совершить

ошибки при воплощении своих решений в коде программного продукта.

Корректному представлению решений в коде могут помешать [5]как обычные опечатки, так и забывчивость программиста или его нежелание отказаться от привычных приемов, которые не дают возможности правильно записать принятое решение. С ошибками такого рода можно справиться при помощи инспектирования кода, взаимного контроля, при котором разработчики внимательно читают код друг друга, а также при помощи опережающей разработки модульных тестов и тестирования.
В программировании баг (bug — жук) — жаргонное слово, которое, как правило, обозначает ошибку в программе или системе, которая выдает неожиданный или неправильный результат. Большинство багов возникают из-за ошибок, [2] совершенных разработчиками программы в её исходном коде, либо в [5]

самой логике программного продукта.

Также [5]некоторые баги
возникают из-за некорректной работы компилятора, вырабатывающего некорректный код. Программу, которая содержит большое число багов и/или баги, [2]серьёзно ограничивающие её [5]функциональность, называют нестабильной или, на жаргонном языке, «глючной», «забагованной» (unstable, buggy).
Термин «баг» обычно употребляется в отношении ошибок, проявляющих себя на стадии работы программы, в отличие, например, от ошибок проектирования или синтаксических ошибок. Отчет, содержащий информацию о баге [2]

могут называть баг-репортом (от англ. bug report – отчет
об ошибке), а также

отчетом об ошибке или отчетом о проблеме. Отчет о
критической проблеме (crash), вызывающей аварийное завершение программы, называют крэш-репортом (crash report). Ошибки и неточности
32
локализуются и устраняются в процессе тестирования и отладки программы [22].
1.2.1 [2]Жизненный цикл дефекта
Рисунок 1.4 иллюстрирует жизненный цикл дефекта, принятый во многих крупных компаниях. Баг может находиться в одном из представленных на рисунке состояний.
1) Обнаружен (submitted). Тестировщик находит [15]

ошибку и
представляет ее на рассмотрение в систему отслеживания ошибок. С этого момента ошибка официально обнаружена и о ней оповещаются разработчики.
2)

Изначен (assigned). Тестировщик или ведущий разработчик рассматривает баг и [15]

назначает задачу по его устранению подходящему
программисту из

команды разработчиков.
3) Исправлен (fixed). Разработчик, которому было назначено исправление дефекта, [15]

устраняет его и сообщает ответственному лицу о

том,
что задание выполнено.
Рисунок 1.4 – Жизненный цикл дефекта
4) Проверен (verified). Тестировщик, который обнаружил ошибку, проверяет [15]

исправленную версию в новой сборке программного продукта. И
33

только в том случае, если ошибка не проявится в новой сборке, тестировщик меняет статус бага на Verified.
5) Открыт заново (reopened). Если [15]

ошибка снова проявляется в новой
сборке, тестировщик снова открывает задачу по устранению этой ошибки.
6)

Отклонен (declined). Баг может быть отклонен. Во-первых, потому что для заказчика [15]

некоторые ошибки могут перестать быть актуальными. Вовторых, это может случиться

по вине тестировщика из-за плохого знания продукта или требований (дефекта на самом деле нет).
7) Отложен (deferred). Если исправление [15]

конкретной ошибки сейчас не
имеет первостепенную задачу, или программист ждет, к примеру, информацию, от которой зависит исправление ошибки, тогда она

приобретает статус Deferred.
8) Закрытые (closed) ошибки. Закрытым считается баг в состояниях Проверен (verified) и Отклонен (declined).
9) Открытые (opeп) баги. Открытыми являются баги в состояниях Обнаружен (submitted), Назначен (assigned), Открыт заново (reopened). Иногда к открытым относят и баги в состояниях Исправлен (fixed) и Отложен (deferred) [23].

1.3 [15] Непрерывная интеграция

Непрерывная интеграция (CI, англ. Continuous Integration) — это практика разработки программного обеспечения, которая заключается в слиянии рабочих копий в общую основную ветвь разработки несколько раз в день и выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем. В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может [2]непредсказуемо [4]задержать окончание работ. Переход к непрерывной интеграции позволяет снизить [2]трудоёмкость интеграции и сделать её более предсказуемой за счет наиболее раннего обнаружения и устранения ошибок и противоречий, [4]

но основным
34
преимуществом является сокращение стоимости исправления дефекта, за счёт раннего его выявления. Непрерывная интеграция впервые названа и предложена Гради Бучем в 1991 г.
Непрерывная интеграция нацелена на ускорение и облегчение процесса выявления проблем, возникающих в процессе разработки программного обеспечения. При регулярной интеграции изменений единовременный объем проверок уменьшается. В результате на отладку тратится меньше времени, оставшееся время можно перераспределить на добавление новых функций. Также возможно добавить проверку стиля кода, цикломатической сложности (чем ниже сложность, тем легче тестировать) и другие виды контроля. Это упрощает рецензирование кода (code review), экономит время и улучшает качество кода [24].
Рисунок 1.5– Схематичное представление непрерывной интеграции

1.3.1 Организация непрерывной интеграции

На выделенном сервере организуется служба, в задачи которой входят:
1) получение исходного кода из репозитория;
2) сборка проекта;
3) выполнение тестов;

4) [2]развёртывание [4]готового проекта;
5) отправка отчетов.
35
Локальная сборка может осуществляться:
1) по внешнему запросу;
2) по расписанию;
3) по факту обновления репозитория и по другим [2]критериям.
[4]Сборка по расписанию:
В [2]случае сборки по расписанию (англ. daily build — рус. ежедневная сборка), они, как правило, проводятся каждой ночью в автоматическом режиме — ночные сборки ([4]чтобы к началу рабочего дня были готовы результаты тестирования). Для [2]различия дополнительно [4]вводится система нумерации [2]сборок — [4]обычно, каждая сборка нумеруется натуральным числом, которое увеличивается с [2]каждой новой сборкой. Исходные тексты и другие исходные данные при взятии их из репозитория (хранилища) системы контроля версий помечаются номером сборки. Благодаря этому, точно такая же [4]сборка может быть точно воспроизведена в будущем — достаточно взять исходные данные по нужной метке и запустить процесс снова. Это [2]даёт [4]возможность повторно выпускать даже очень старые версии программы с небольшими исправлениями.
[2]Преимущества:
1) проблемы интеграции выявляются и исправляются быстро, что оказывается дешевле;
2) [4]немедленный прогон модульных тестов для свежих изменений;
3) постоянное наличие текущей стабильной версии вместе с продуктами сборок — для тестирования, демонстрации, и т. п;
4) немедленный эффект от неполного или неработающего кода приучает разработчиков к работе в итеративном режиме с более коротким циклом.
Недостатки:
1) [2]затраты на поддержку работы непрерывной интеграции;
2) потенциальная необходимость в выделенном сервере под нужды непрерывной интеграции;
36
3) [4]немедленный эффект от неполного или неработающего кода отучает разработчиков от выполнения периодических резервных включений кода в репозиторий.
4) в [2]случае использования системы управления версиями исходного кода с поддержкой ветвления, эта проблема может решаться созданием отдельной «ветки» (англ. branch) проекта для внесения крупных изменений (код, разработка которого до работоспособного варианта займет несколько дней, но желательно более частое сохранение результата в репозиторий). [4]По окончании разработки и индивидуального тестирования такой ветки, она может быть объединена ([2]англ. merge) с [4]основным кодом или «стволом» ([2]англ. trunk) проекта.
1.3.2 [4]

Популярные CI-платформы.
1.3.2.1 CircleCI
Функции:
1) CircleCI — это облачная система, для которой не нужно настраивать отдельный сервер и которую не придется администрировать. Однако существует и локальная версия, которую можно развернуть в частном облаке.
2) даже для коммерческого использования существует бесплатная версия.
3) с помощью REST API можно получить доступ к проектам, сборкам и артефактам. Результатом сборки является артефакт или группа артефактов. Артефактом могут быть скомпилированное приложение или исполняемые

файлы (например, APK для Android) или метаданные (например, информация
об удачно завершившемся тестировании).
4) CircleCI кэширует сторонние зависимости, что позволяет избежать
постоянной установки необходимых окружений.
5) существует возможность подключения к контейнеру по SSH. Это
может потребоваться, если возникнут какие-то проблемы.
37
6) CircleCI — полностью готовое решение, требующее минимальной
настройки.
Совместимость с:
Python, Node.js, Ruby, Java, Go и т. д.;
1) Ubuntu (12.04, 14.04), Mac OS X (платные аккаунты);
2) Github, Bitbucket;
3) AWS, Azure, Heroku, Docker, выделенный сервер;
4) Jira, HipChat, Slack.
Достоинства CircleCI:
1) легкое и быстрое начало работы;
2) бесплатная версия для коммерческого использования;
3) небольшие и легко читаемые файлы конфигурации в формате
YAML;
4) отсутствие необходимости в выделенном сервере CircleCI.
Недостатки CircleCI:
1) CircleCI в бесплатной версии поддерживает только Ubuntu 12.04 и
14.04. Для использования MacOS придется заплатить;
2) несмотря на то что CircleCI может работать с любыми языками
программирования, в базовой комплектации поддерживаются только Go
(Golang), Haskell, Java, PHP, Python, Ruby/Rails, Scala;
3) При желании подстроить систему под себя в некоторых случаях
могут возникнуть проблемы, и тогда для достижения цели понадобится
стороннее программное обеспечение [25].
1.3.2.2 Travis CI
Travis CI:
1) использует файлы конфигурации в формате YAML;
2) развернута в облаке;
3) поддерживает Docker для запуска тестов.
Что есть в TravisCI :
1) запуск тестов одновременно под Linux и Mac OS X.
38
2) поддержка большего количества языков (в базовой комплектации):
Android, C, C#, C++, Clojure, Crystal, D, Dart, Erlang, Elixir, F#, Go, Groovy,
Haskell, Haxe, Java, JavaScript (with Node.js), Julia, Objective-C, Perl, Perl6,
PHP, Python, R, Ruby, Rust, Scala, Smalltalk, Visual Basic.
3) поддержка build matrix.
Достоинства Travis CI:
1) Build matrix — это инструмент, который дает возможность
выполнять тесты, используя разные версии языков и пакетов. Он обладает
богатыми возможностями по настройке. Например, при неудачных сборках в
некоторых окружениях система может выдать предупреждение, но сборка
целиком не будет считаться неудачной (это удобно при использовании devверсий пакетов);
2) быстрый старт;
3) небольшие и легко читаемые файлы конфигурации в формате
YAML;
4) бесплатная версия для opensource-проектов;
5) отсутствие необходимости в выделенном сервере.
Недостатки Travis CI:
1) по сравнению с CircleCI цены выше, нет бесплатной версии для
коммерческого использования;
2) ограниченные возможности по настройке (для некоторых вещей
может потребоваться сторонний софт) [26].
1.3.2.3 Jenkins
Возможности:
1) Jenkins — это автономное приложение на Java, которое может
работать под Windows, Mac OS X и другими unix-подобными

операционными системами.
2) в Update Center можно найти сотни плагинов, поэтому Jenkins интегрируется практически с любым инструментом, относящимся к непрерывной интеграции и непрерывной поставке (continuous delivery).
39
3) возможности Jenkins могут быть практически неограниченно расширены благодаря системе подключения плагинов.
4) предусмотрены различные режимы: Freestyle project, Pipeline, External Job, Multi-configuration project, Folder, GitHub Organization, Multibranch Pipeline.
5) Jenkins Pipeline — это набор плагинов, поддерживающих создание и интеграцию в Jenkins цепочек непрерывной поставки. Pipeline предоставляет расширяемый набор инструментов по моделированию цепочек поставки типа "as code" различной степени сложности с помощью Pipeline DSL.
6) позволяет запускать сборки с различными условиями.
7) Jenkins может работать с Libvirt, Kubernetes, Docker и др.
8) используя REST API, можно контролировать количество получаемых данных, получать/обновлять config.xml, удалять задания (job), получать все сборки, получать/обновлять описание задания, выполнять сборку, включать/отключать задания.
Достоинства Jenkins:
1) цена (он бесплатен);
2) возможности по настройке;
3) система плагинов;
4) полный контроль над системой.
Недостатки Jenkins:
1) требуется выделенный сервер (или несколько серверов), что влечет за собой дополнительные расходы на сам сервер, DevOps и т. д.;
2) на настройку необходимо время [27].

1.4 Фреймворки тестирования

1.4.1 xUnit

xUnit — это собирательное название семейства фреймворков для модульного тестирования, структура и функциональность которых основана на [7]

SUnit, который предназначался для языка программирования Smalltalk.
40
SUnit, разработанный

Кентом Беком в 1998 году, был написан в высоко структурном объектно-ориентированном стиле, получил широкую популярность и был адаптирован для множества других языков. Названия фреймворков этого семейства образованы аналогично "SUnit", обычно заменяется буква "S" на первую букву (или несколько первых) в названии предполагаемого языка (" JUnit" для Java, "NUnit" для программной платформы .NET и т. д.). Семейство таких фреймворков с общей архитектурой обычно и известно, как "xUnit".
Архитектура xUnit
Все фреймворки из семейства xUnit имеют следующие базовые компоненты архитектуры, которые в различных реализациях могут слегка варьироваться.
Модуль, выполняющий тестирование (Test runner)
Модуль представляет собой исполняемую программу, которая выполняет тесты, реализованные с помощью фреймворка, и отображает информацию о ходе их выполнения.
Тестовые сценарии (Test cases)
Варианты тестирования (тестовые сценарии/случаи) являются базовыми элементами модульных тестов.
Конфигурации тестирования (Test fixtures)
Конфигурация тестирования (также называемая контекстом) — это набор предварительно заданных условий или состояний объектов,

необходимый для запуска теста. Разработчик должен задать заведомо корректную конфигурацию перед выполнением каждого теста, а затем вернуть оригинальную конфигурацию после завершения теста.
Наборы тестов (Test suites)
Тестовый набор — это несколько тестов, имеющих общую конфигурацию. Очерёдность выполнения тестов не должна иметь значения.
Выполнение тестов (Test execution)
Выполнение каждого теста происходит согласно следующей схеме:
41
setup(); /* Сначала подготавливается 'контекст' тестирования */
...
/* Тело теста - здесь указывается тестовый сценарий */
...
teardown(); /* После прохождения теста (независимо от его результата) контекст тестирования "очищается" */
Форматирование результатов тестирования (Test result formatter)
Модуль, выполняющий тестирование, должен вывести результаты в одном или нескольких заданных форматах. В дополнение к обычному тексту, воспринимаемому человеком, часто результаты выводятся в формате XML.
Утверждения (Assertions)
Утверждение в тесте — это функция или макрос, которая проверяет поведение или состояние тестируемого модуля. Часто утверждением является проверка равенства или неравенства некоторого параметра модуля ожидаемому результату. Неудачное прохождение проверки приводит к провалу всего тестового сценария и к исключению (если оно необходимо), которое останавливает сценарий без перехода к следующему утверждению.
Фреймворки xUnit
Фреймворки с архитектурой, характерной для xUnit, существуют для множества языков программирования и платформ разработки. Примеры:
1) [7]

CppUnit - фреймворк для C++;
2) DUnit - инструмент для среды разработки Delphi;
3) JUnit - библиотека для Java;
4) NUnit, xUnit.NET -

среда юнит-тестирования для .NET;
5) phpUnit - библиотека для [7] PHP [28].
1.4.2 NUnit
NUnit — открытая среда юнит- тестирования приложений для .NET. Она была [19]портирована с [22]языка Java (библиотека JUnit). Первые версии NUnit
42
были написаны на J#, но затем весь код был переписан на C# с использованием таких новшеств .NET, как атрибуты.
Существуют также известные расширения оригинального пакета NUnit, большая часть которых представлена с открытым исходным кодом. NUnit.Forms дополняет NUnit средствами тестирования элементов пользовательского интерфейса Windows Forms. NUnit.ASP выполняет ту же задачу для элементов интерфейса в ASP.NET.
[19]

Особенности
1) Тесты можно запускать из консоли, в Visual Studio через тестовый адаптер или через сторонние приложения;
2) Тесты могут выполняться параллельно;
3) Сильная поддержка тестов, управляемых данными;
4) Поддерживает несколько платформ, включая .NET Core, Xamarin Mobile, Compact Framework и Silverlight;
5) Каждый тестовый пример может быть добавлен в одну или несколько категорий, чтобы обеспечить избирательный запуск.
NUnit предоставляет консольное приложение (nunit3-console.exe),

которое используется для пакетного выполнения тестов. Коннектор работает через NUnit Test Engine, который предоставляет ему возможность загружать, исследовать и выполнять тесты. Когда тесты должны запускаться в отдельном процессе, движок использует программу nunit-agent для их запуска.

Пример

```
using NUnit.Framework;
[TestFixture]
public class ExampleTestOfNUnit
{
[Test]
public void TestMultiplication()
43
{
Assert.AreEqual(4, 2*2, "Multiplication");
// Equivalently, since version 2.4 NUnit offers a new and
// more intuitive assertion syntax based on constraint objects
// [http://www.nunit.org/index.php?p=constraintModel&r=2.4.7]:
Assert.That(2*2, Is.EqualTo(4), "Multiplication constraint-based");
}
}
// The following example shows different ways of writing the same exception test.
[TestFixture]
public class AssertThrowsTests
{
[Test]
public void Tests()
{
// .NET 1.x
Assert.Throws(typeof(ArgumentException),
new TestDelegate(MethodThatThrows));
// .NET 2.0
 AssertThrows< ArgumentException>(MethodThatThrows);
Assert.Throws<ArgumentException>(
delegate { throw new ArgumentException(); });
// Using C# 3.0
Assert.Throws<ArgumentException>(
() => { throw new ArgumentException(); });
}
void MethodThatThrows()
44
{
throw new ArgumentException();
}
}
// This example shows use of the return value to perform additional verification of
the exception.
[TestFixture]
public class UsingReturnValue
{
[Test]
public void TestException()
{
MyException ex = Assert.Throws<MyException>(
delegate { throw new MyException("message", 42); });
Assert.That(ex.Message, Is.EqualTo("message"));
Assert.That(ex.MyParam, Is.EqualTo(42));
}
```

```
}
```
[13]

Расширения
1) FireBenchmarks – это расширение, способное записывать время выполнения модульных тестов и генерировать отчеты о производительности XML, CSV, XHTML с диаграммами и отслеживанием истории. Его основная цель – предоставить разработчику или команде, которая работает с гибкой методологией для ��нтеграции показателей производительности и анализа, единую тестовую среду для легкого контроля и мониторинга эволюции программной системы с точки зрения сложности алгоритмической нагрузки и загрузки системных ресурсов.
2) NUnit.Forms – это расширение базовой структуры NUnit с открытым исходным кодом. В нем конкретно рассматривается расширение NUnit,
45
чтобы иметь возможность обрабатывать элементы пользовательского интерфейса в Windows Forms.
3) NUnit.ASP является прекращенным расширением к основной структуре NUnit и также имеет открытый исходный код. Он специально рассматривает расширение NUnit, чтобы иметь возможность обрабатывать элементы пользовательского интерфейса в ASP.Net [29].
1.4.3 Visual Studio Unit Testing Framework
Visual Studio Unit Testing Framework интегрирован в версии Visual Studio 2005 и более поздних версий. Структура тестирования модулей определена в Microsoft.VisualStudio.QualityTools.UnitTestFramework.dll. Модульные тесты, созданные с помощью модуля тестирования, могут быть выполнены в Visual Studio или могут быть запущены с помощью MSTest.exe с параметрами из командной строки.
MSTest.exe — это запускаемая из командной строки программа, которая служит для запуска тестов. У этой программы есть несколько параметров, с помощью которых можно настроить ее работу. Их можно указать в командной строке MSTest.exe в произвольном порядке.
Элементы тестирования:
Класс тестирования (Test class)
Тестовые классы объявляются с помощью атрибута TestClass. Атрибут используется для идентификации классов, содержащих методы тестирования.
Метод тестирования (Test method)
Методы тестирования объявляются атрибутом TestMethod. Атрибут используется для идентификации методов, которые содержат единичный тестовый код.
Утверждения (Assertions)
Утверждение представляет собой фрагмент кода, который выполняется, чтобы проверить состояние или поведение в отношении ожидаемого результата. Утверждения в модульном тестировании Visual Studio выполняются путем вызова методов в классе Assert.
46
Инициализация и методы очистки (Initialization and cleanup methods)
Методы инициализации и очистки используются для подготовки модульных тестов перед запуском и очисткой после выполнения модульных тестов. Методы инициализации объявляются путем добавления атрибута TestInitialize, тогда как методы очистки объявляются путем добавления атрибута TestCleanup [30].
1.5 Обзор систем контроля версий

Системы контроля версий стали неотъемлемой частью жизни не только разработчиков программного обеспечения, но и всех людей, столкнувшихся с проблемой управления интенсивно изменяющейся информацией, и желающих облегчить себе жизнь. Вследствие этого, появилось большое число различных продуктов, предлагающих широкие возможности и предоставляющих обширные инструменты для управления версиями. В этой статье будут кратко рассмотрены наиболее популярные из них, приведены их

достоинства и недостатки.
Для сравнения были выбраны наиболее распространенные системы контроля версий: [21]

Subversion, Git, Mercurial [31].
1.5.1 Система управления версиями Subversion.
SVN создавалась как альтернатива CVS с целью исправить недостатки CVS и в то же время обеспечить высокую совместимость с ней.
Как и CVS, SVN это бесплатная система контроля версий с открытым исходным кодом. С той лишь разницей, что распространяется под лицензией Apache, а не под Открытым лицензионным соглашением GNU.
Для сохранения целостности базы данных SVN использует так называемые атомарные операции. При появлении новой версии к финальному продукту применяются либо все исправления, либо ни одно из них. Таким образом, код защищают от хаотичных частичных правок, которые не согласуются между собой и вызывают ошибки.
Многие разработчики переключились на SVN, так как новая
47
технология унаследовала лучшие возможности CVS и в то же время расширила их.
В то время как в CVS операции с ветками кода дорогостоящие и не предусмотрены архитектурой системы, SVN создана как раз для этого. То есть, для более крупных проектов с ветвлением кода и многими направлениями разработки.
В качестве недостатков SVN упоминаются сравнительно низкая скорость и нехватка распределенного управления версиями. Распределенный контроль версий использует пиринговую модель, а не централизованный сервер для хранения обновлений программного кода. И хотя пиринговая модель работает лучше в opeп source проектах, она не идеальна в других случаях. Недостаток серверного подхода в том, что когда сервер падает, то у клиентов нет доступа к коду.
Достоинства:
1) разнообразные графические интерфейсы и удобная работа из консоли;
2) отслеживается история изменения файлов и каталогов даже после их переименования и перемещения;
3) высокая эффективность работы, как с текстовыми, так и с бинарными файлами;
4) встроенная поддержка во многие интегрированные средства разработки, такие как KDevelop, Zend Studio и многие другие;
5) возможность создания зеркальных копий репозитория;
6) два типа репозитория – база данных или набор обычных файлов;
7) возможность доступа к репозиторию через Apache с использованием протокола WebDAV;
8) наличие удобного механизма создания меток и ветвей проектов;
9) можно с каждым файлом и директорией связать определенный набор свойств, облегчающий взаимодействие с системой контроля версии;
10) широкое распространение позволяет быстро решить большинство
48
возникающих проблем, обратившись к данным, накопленным Интернетсообществом.
Недостатки:
1) полная копия репозитория хранится на локальном компьютере в скрытых файлах, что требует достаточно большого объема памяти;
2) существуют проблемы с переименованием файлов, если переименованный локально файл одним клиентом был в это же время изменен другим клиентом и загружен в репозиторий;
3) слабо поддерживаются операции слияния веток проекта;
4) сложности с полным удалением информации о файлах, попавших в репозиторий, так как в нем всегда остается информация о предыдущих изменениях файла, и непредусмотрено никаких штатных средств для полного удаления данных о файле из репозитория [32].
1.5.2 Система управления версиями Git.
С февраля 2002 года для разработки ядра Linux’а большинством

программистов стала использоваться система контроля версий BitKeeper. Довольно долгое время с ней не возникало проблем, но в 2005 году Лари МакВоем (разработчик BitKeeper'а) отозвал бесплатную версию программы. Разрабатывать проект масштаба Linux без мощной и надежной системы контроля версий – невозможно. Одним из кандидатов и наиболее подходящим проектом оказалась система контроля версий Monotine, но Торвальдса Линуса не устроила ее скорость работы. Так как особенности организации Monatone не позволяли значительно увеличить скорость обработки данных, то 3 апреля 2005 года Линус приступил к разработке собственной системы контроля версий – Git.
Практически одновременно с Линусом (на три дня позже), к разработке новой системы контроля версий приступил, и Мэтт Макал. Свой проект Мэтт назвал Mercurial, но об этом позже, а сейчас вернемся к распределенной системе контроля версий Git.

Git – это гибкая, распределенная (без единого сервера) система
49
контроля версий, дающая массу возможностей не только разработчикам программных продуктов, но и писателям для изменения, дополнения и отслеживания изменения «рукописей» и сюжетных линий, и учителям для корректировки и развития курса лекций, и администраторам для ведения документации, и для многих других направлений, требующих управления историей изменений.
У [18]каждого разработчика, использующего Git, есть свой локальный репозиторий, позволяющий локально управлять версиями. Затем, сохраненными в локальный репозиторий данными, можно обмениваться с другими пользователями.
Часто при работе с Git создают центральный репозиторий, с которым остальные разработчики синхронизируются. Пример организации системы с центральным репозиторием – это проект разработки ядра Linux'а.
В [10]

этом случае все участники проекта ведут свои локальны разработки и беспрепятственно скачивают обновления из центрального репозитория. Когда необходимые работы отдельными участниками проекта выполнены и отлажены, они, после удостоверения владельцем центрального репозитория в корректности и актуальности проделанной работы, загружают свои изменения в центральный репозиторий.
Наличие локальных репозиторием также значительно повышает надежность хранения данных, так как, если один из репозиториев выйдет из строя, данные могут быть легко восстановлены из других репо��иториев.
Работа над версиями проекта в Git может вестись в нескольких ветках, которые затем могут с легкостью полностью или частично объединяться, уничтожаться, откатываться и разрастаться во все новые и новые ветки проекта.
Можно долго обсуждать возможности Git'а, но для краткости и более простого восприятия приведем основные достоинства и недостатки этой системы управления версиями.
Достоинства:
50
1)

надежная система сравнения ревизий и проверки корректности данных, основанные на алгоритме хеширования (Secure Hash Algorithm 1);
2) гибкая система ветвления проектов и слияния веток между собой;
3) наличие локального репозитория, содержащего полную информацию обо всех изменениях, позволяет вести полноценный локальный контроль версий и заливать в главный репозиторий только полностью прошедшие проверку изменения;
4) [10]высокая производительность и скорость работы;
5) удобный и интуитивно понятный [17] набор команд;

6) множество графических оболочек, позволяющих быстро и качественно вести работы с Git'ом;
7) возможность делать контрольные точки, в которых данные сохраняются без дельты компрессии, а полностью. Это позволяет уменьшить скорость восстановления данных, так как за основу берется ближайшая контрольная точка, и восстановление идет от нее. Если бы контрольные точки отсутствовали, то восстановление больших проектов могло бы занимать часы;
8) широкая распространенность, легкая доступность и качественная документация;
9) [10]

гибкость системы позволяет удобно ее настраивать и даже создавать специализированные контроля

системы или пользовательские интерфейсы на базе git;
10) [17]универсальный сетевой доступ с использованием протоколов http, ftp, [10]rsync, [17]ssh и др.
Недостатки:
1) Unix – ориентированность. На данный момент отсутствует зрелая реализация Git, совместимая с другими операционными системами;
2) возможные (но чрезвычайно низкие) совпадения хеш - кода отличных по содержанию ревизий.3. Не отслеживается изменение отдельных файлов, а только всего проекта целиком, что может быть
51
неудобно при работе с большими проектами, содержащими множество несвязных файлов;
3) при начальном (первом) создании репозитория и синхронизации его с другими разработчиками, потребуется достаточно длительное время для скачивания данных, особенно, если проект большой, так как требуется скопировать на локальный компьютер весь репозиторий [33].
1.5.3 [10]Система управления версиями Mercurial.
Распределенная система контроля версий Mercurial разрабатывалась Мэттом Макалом параллельно с системой контроля версий Git, созданной Торвальдсом Линусом.
Первоначально, она была создана для эффективного управления большими проектами под Linux'ом, а поэтому была ориентирована на быструю и надежную работу с большими репозиториями. На данный момент mercurial адаптирован для работы [14]под Windows, Mac OS X и большинство Unix систем.
[10]Большая часть системы контроля версий написана на языке Python, и только отдельные участки программы, требующие наибольшего быстродействия, написаны на языке Си.
[14]

Идентификация ревизий происходит на основе алгоритма хеширования SHA1 (Secure Hash Algorithm 1), однако, также предусмотрена возможность присвоения ревизиям индивидуальных номеров.
Так же, как и в git'е, поддерживается возможность создания веток проекта с последующим их слиянием.
Для взаимодействия между клиентами используются протоколы HTTP, HTTPS или SSH.

Набор команд - простой и интуитивно понятный, во многом схожий с командами subversion. Так же имеется ряд графических оболочек и доступ к репозиторию через веб-интерфейс. Немаловажным является и наличие утилит, позволяющих импортировать репозитории многих других систем контроля версий.
52
Достоинства:

1) быстрая обработка данных;
2) [14]кроссплатформенная поддержка;
3) [17]возможность работы с несколькими ветками проекта;
4) [14] простота в обращение;
5) возможность конвертирования репозиториев других систем поддержки версий, таких как CVS, Subversion, Git, Darcs, GNU Arch, Bazaar и др.

[17]Недостатки:
1) [14]возможные (но чрезвычайно низкие) совпадения хеш - кода отличных по содержанию ревизий;
2) [10]ориентирован на работу в консоли [34].

1.6 [14]

TestLink

TestLink - это веб - система управления тестированием, которая облегчает обеспечение качества программного продукта. TestLink разработан и поддерживается Teamtest. Платформа предлагает поддержку тестовых примеров, наборов тестов, планов тестирования, тестовых проектов и управления пользователями, а также различных отчетов и статистических данных [36].

Рисунок 1.6 – Форма входа в TestLink

53

1.6.1 Использование

Основными единицами, используемыми TestLink, являются: тестовый пример, тестовый комплект, план тестирования, тестовый проект и пользователь.

План тестирования

Тестовые планы - это базовая единица для выполнения набора тестов в приложении. Планы испытаний включают сборку, назначение пользователя и результаты испытаний.

План тестирования содержит имя, описание, коллекцию выбранных тестовых случаев, сборки, результаты тестирования, назначение тестировщика и определение приоритета. Каждый план тестирования связан с текущим проектом тестирования.

Планы тестирования могут быть созданы на странице «Управление планом тестирования» пользователями с ведущими привилегиями для текущего проекта тестирования.

Определение плана тестирования состоит из заголовка, описания (htmlформат) и статуса «Активный». Описание должно включать следующую информацию в отношении процессов компании:
1) проверяемые характеристики;
2) особенности, которые нельзя тестировать;
3) критерии испытаний (для прохождения тестируемого продукта);
4) тестовая среда, Инфраструктура;
5) инструменты тестирования;
6) риски.

Планы испытаний состоят из тестовых случаев, импортированных из тестовой спецификации в определенный момент времени. Планы испытаний могут быть созданы из других тестовых планов. Это позволяет пользователям создавать тестовые планы из тестовых случаев, которые существуют в нужный момент времени. Это может потребоваться при создании тестового плана для патча. Чтобы пользователь мог видеть План

54

тестирования, они должны обладать надлежащими правами. Права могут быть назначены (по указанию) в разделе «Определить права пользователя / проекта». Это важно помнить, когда пользователи говорят, что они не могут видеть проект, над которым они работают.

Рисунок 1.7 – Пример тестового плана

Тестовый пример

В тестовом примере описывается простая задача в рабочем процессе приложения. Тест является фундаментальной частью TestLink. Тестовые примеры организованы в тестовые комплекты:

1) идентификатор тестового примера автоматически назначается TestLink и не может быть изменен пользователями. Этот идентификатор состоит из префикса Test Project и счетчика, связанного с тестовым проектом, в котором создан тестовый пример;
2) название: может содержать краткое описание или аббревиатуру (например, TL-USER-LOGIN);
55
3) резюме: должно быть очень короткое; просто для обзора, введения и ссылок;
4) шаги: описание тестового сценария (входные действия); может также включать предварительное условие и информацию для очистки здесь;
5) ожидаемые результаты: описать контрольные точки и ожидаемое поведение тестируемого продукта или системы;
6) вложения: могут быть добавлены, если конфигурация позволяет это;
7) важность: Дизайнер тестов мог установить важность теста [HIGH, MEDIUM и LOW];
8) тип выполнения: Дизайнер тестов мог установить поддержку автоматизации теста [MANUAL / AUTOMATED];
9) пользовательские поля: администратор может определять собственные параметры для улучшения описания тестового примера или категоризации. Большие пользовательские поля (более 250 символов) невозможны. Но информация может быть добавлена в родительский тестовый набор и передаваться через настраиваемые поля. Например, вы можете описать «Стандарт», «Производительность», «Стандарт_2» и «Ссылаться через CF» на эти метки.
Рисунок 1.8 – Изображение тестового примера
Пользователь
56
Каждый пользователь TestLink имеет назначенную роль, которая определяет доступные функции. Типы по умолчанию: Guest, Test Designer, Senior tester, Tester, Leader и Administrator, но также могут быть созданы пользовательские роли.
Тестовые проекты
Тестовые проекты - это основная организационная единица TestLink. Тестовые проекты могут быть продуктами или решениями вашей компании, которые могут менять свои функции и функциональность с течением времени, но по большей части остаются теми же. Проект тестирования включает в себя документацию по требованиям, спецификацию тестирования, тестовые планы и конкретные права пользователей. Проекты тестирования независимы и не делят данные.
Технические характеристики теста
TestLink разбивает структуру тестовых спецификаций на тестовые объекты и тестовые случаи. Эти уровни сохраняются во всех приложениях. Один тестовый проект имеет только одну тестовую спецификацию [36].
1.6.2 Особенности
1) пароли пользователей и управление ими;
2) группировка тестовых примеров в тестовых спецификациях;
3) планы испытаний;
4) платформы;
5) требования к версии и ревизии;
6) поддержка тестирования различных сборок программного обеспечения;
7) отчеты, графики и мониторы;
8) настройка пользовательского интерфейса с использованием шаблонов Smarty;
9) интеграция с LDAP;
10) интеграция с другим программным обеспечением с использованием предоставленного API;
57
11) интеграция системы отслеживания ошибок (Mantis, JIRA, Bugzilla, FogBugz, Redmine и другие).
58
Выводы по первой главе

В первой главе были рассмотрены теоретические аспекты процесса тестирования. Определено понятие тестирование программного продукта.
Кроме того, были выделены основные классификации видов тестирования:
1) по объекту тестирования;
2) тестирование, связанное с изменениями;
3) по уровню тестирования;
4) по исполнению кода;
5) по субъекту тестирования;
6) по позитивности сценария;
7) по степени автоматизации.
Также были изучены различные методологии тестирования:
1) метод черного ящика;
2) метод белого ящика;
3) метод серого ящика.
Изучены аспекты разработки и выполнения различных тест-кейсов и проанализированы основные атрибуты, которые включает в себя любой тесткейс. Также изучены требования, которые применяются при составлении тест-кейсов. Кроме того, были описаны и проанализированы различные виды дефектов, которые могут присутствовать в любом ПО и был изучен жизненный цикл любого дефекта.
Проведен анализ популярных CI платформ, таких как CircleCI, Travis CI, Jenkins. Выявлены достоинства и недостатки данных систем непрерывной интеграции и была представлена схема непрерывной интеграции при использовании CI платформ.
Проанализированы различные фреймворки тестирования: xUnit, NUnit, Visual Studio Unit Testing Framework, а также были выявлены их достоинства и недостатки.
59
Далее были изучены системы контроля версий, такие как: Subversion, Git, Mercurial. Были проанализированы их достоинства и недостатки, а также способы работы с ними.
После этого была рассмотрена система TestLink. Так как основная часть тестов для ручного тестирования на предприятии размещена в данной системе. Были изучены способы работы с данной системой, а также ее достоинства и недостатки.
60
Глава 2. Экспериментальная часть
2.1 Alpha.Alarms
Приложение Alpha.Alarms используется в пунктах автоматизации и мониторинга технологических процессов. Применяется для отслеживания событий и тревог, которые появляются при изменении состояний технологических объектов. Основные функции приложения:
1) отображение сообщений о событиях и тревогах в режиме реального времени (оперативный режим);
2) отображение истории сообщений о событиях и тревогах за прошедшие периоды (исторический режим).
Alpha.Alarms может использоваться как встраиваемый компонент или работать как самостоятельное приложение. В качестве встраиваемого компонента Alpha.Alarms может использоваться в Alpha.HMI, а также в HMI SCADA систем сторонних разработчиков - Genesis, InfinitySCADA, iFix и других [37].
2.1.1 Системные требования
1) Операционная система: MS Windows 7/2008 Server;
2) Разрядность ОС: x64 или x32;
3) Процессор: Intel Celeron 1.6 ГГц и выше;
4) Объем оперативной памяти: не менее 1 ГБ;
5) Объем дисковой памяти: не менее 500 МБ;
6) Сетевой адаптер: Ethernet 10/100/1000 Мбит/с;
2.1.2Варианты запуска Alpha.Alarms
2.1.2.1 Стандартный запуск
Для стандартного запуска Alpha.Alarms, как отдельного приложения, воспользуйтесь командой Пуск Все программы Automiq Alpha.Alarms Alpha.Alarms или Пуск Все программы Automiq

Alpha.Alarms Alpha.Alarms (x64). Аналогичное действие можно выполнить, запустив исполняемый файл Alpha.Alarms.App.exe,
61
расположенный в папке C:\Program Files (x86)\Automiq\Alpha.Alarms или C:\Program files\Automiq\Alpha.Alarms.

2.1.2.2 Запуск с параметрами

Использование параметров при запуске Alpha.Alarms позволяет пользователю задавать предварительные установки конфигурационных данных, заменяющие настройки по умолчанию. Для запуска приложения с параметрами используется командная строка, вызываемая командой cmd в строке поиска, либо с помощью команды Пуск Все программы Стандартные Командная строка.

2.1.2.3 Запуск Alpha.Alarms как встраиваемого компонента

Для внедрения встраиваемого компонента Alpha.Alarms в существующий проект автоматизации необходимо добавить новый объект типа Alpha.Alarms в приложение-контейнер. Принцип добавления встраиваемого компонента может различаться в зависимости от приложенияконтейнера. В качестве контейнера может выступать приложение Alpha.HMI, а также HMI SCADA систем сторонних разработчиков - Genesis, InfinitySCADA, iFix и других. Ниже приведено несколько примеров добавления компонента Alpha.Alarms.

2.1.3 Просмотр событий

2.1.3.1 Просмотр оперативных событий

Оперативный режим предназначен для поступления оповещений о событиях в режиме реального времени. Для перехода в оперативный режим предназначена кнопка (Оперативный режим) на панели инструментов или аналогичная команда контекстного меню. Сообщения о событиях отображаются в режиме журнала, в котором каждому событию отводится отдельная строка или в режиме отображения только активных событий. Вид окна приложения, отображающего оперативные события, приведен на рисунке ниже.
62
Рисунок 2.1– Оперативный режим

Способ отображения сообщений о событиях зависит от важности событий. Важность событий принимает значения от 1 до 1000. Любое событие может относиться к одной из трех стандартных групп важности:

1) важные;
2) значительные (важность по умолчанию: от 334 до 666);
3) особой важности (важность по умолчанию: от 667 до 1000).

2.1.3.1.1 Квитирование событий

Квитированием события называется отметка о прочтении сообщения о событии, которая выставляется пользователем Alpha.Alarms и фиксируется на стороне AE-сервера вместе со служебной информацией о факте квитирования.

Способы квитирования:

1) выделите одно или несколько событий в таблице и нажмите кнопку (Квитировать) на панели инструментов или в контекстном меню.
3) повторно нажмите левой кнопкой мыши по выделенному событию (способ работает, если установлен флаг квитировать события повторным кликом по выделенной строке в окне Параметры).
4) нажмите на кнопку квитировать в столбце Квитировать
63
5) квитируйте все отображаемые события кнопкой (Квитировать все) на панели инструментов или аналогичной командой в контекстном меню.

2.1.3.1.2 Приостановка поступления событий

Чтобы временно приостановить поступление новых событий в таблицу, включите режим Снимок кнопкой (Снимок) на Панели инструментов или аналогичной командой контекстного меню. При переводе программы в данный режим вновь приходящие сообщения не отображаются, при этом продолжается поступление уведомлений в память приложения и проигрывание звуков поступивших сообщений. Все сообщения, пришедшие

во время активности режима Снимок, отобразятся после его отключения.
2.1.3.1.3 Очистка списка событий
Чтобы очистить список событий оперативного режима, воспользуйтесь кнопкой (Очистить список) на пане��и инструментов либо аналогичной командой контекстного меню.
2.1.3.2 Просмотр истории событий
Исторический режим предназначен для просмотра событий за прошедший период. Для перехода в исторический режим нажмите кнопку (Исторический режим) на панели инструментов или аналогичную команду контекстного меню.
Для просмотра истории событий нужно установить временной интервал выборки данных, выбрать хронологию запрашиваемых данных и запросить данные у источника.
2.1.3.2.1 Установка временного интервала для запроса
Для установки временного интервала, за который требуется запросить данные, следует задать начальное и конечное значение интервала в полях ввода на панели инструментов. Формат даты DD.MM.YYYY hh:mm:ss.
64
Рисунок 2.2 –Установка временного интервала
Установка даты производится путем ввода значений с клавиатуры либо с помощью встроенного календаря. Календарь открывается при нажатии кнопок , расположенных рядом с полем ввода даты.
2.1.4 Дополнительные возможности
2.1.4.1 Фильтрация сообщений о событиях
Для исключения лишних событий из таблицы воспользуйтесь фильтрацией по различным критериям.
Типы фильтрации на стороне источника данных:
1) фильтрация оперативных событий на стороне сервера применяется для ограничения объема отправляемых OPC AE-сервером оперативных событий.
2) фильтр запроса - применяется в момент запроса исторических данных, что уменьшает объем результирующей выборки, поступающей от источника исторических данных.
Типы фильтрации на стороне приложения:
1) фильтр отображения применяется для скрытия лишних событий, присутствующих в данный момент в таблице. Может применяться как в оперативном, так и историческом режиме.
2) фильтрация по объекту срабатывания применяется для отображения оперативных событий определенных технологических объектов.
По форме установки фильтры могут быть:
1) предустановленными - хранятся в приложении для многократного использования.
2) пользовательскими - задаются для использования в рамках текущего сеанса работы с приложением.
65
2.1.4.2 Сохранение данных в табличный файл
В приложении предусмотрена возможность сохранения сообщений, отображаемых в главном окне, в табличный файл. Для сохранения сообщений в табличный файл предназначена кнопка (Сохранить…), расположенная на панели инструментов, а также аналогичная команда в контекстном меню. Возможность сохранения недоступна если таблица сообщений пуста.
Сохранение таблицы сообщений производится в форматах XLSX, XML, CSV.
Для сохранения в табличный файл дополнительной информации (колонтитул, дата печати и т.д.) отметьте флагами нужные элементы в группе Содержание документа узла настроек Печать.
Рисунок 2.3 – Сохранение данных в табличный файл
2.1.4.3 Импорт/экспорт предустановленных фильтров
Чтобы сохранить предустановленные фильтры в XML-файл, перейдите в узел Предустановленные фильтры окна Параметры и нажмите кнопку (Экспорт фильтров...).
Чтобы импортировать предустановленные фильтры из XML-файла,

нажмите (Импорт фильтров...). Импортируемый фильтр добавится в список предустановленных.

66

Рисунок 2.4– Импорт/экспорт предустановленных фильтров

2.1.4.4 Настройка папок для импорта и экспорта

Чтобы указать папки, которые будут предлагаться по умолчанию для операций импорта/экспорта, перейдите в узел Экспорт окна Параметры и укажите пути к папкам для разных категорий файлов.

Рисунок 2.5– Настройка папок для импорта и экспорта

2.1.4.5 Печать таблицы событий

Для печати таблицы событий, предназначена кнопка (Печать…), расположенная на панели инструментов или аналогичная команда в контекстном меню.

67

Рисунок 2.6 – Печать таблицы событий

2.1.4.6 Откат настроек приложения

Для отмены последних настроек, произведенных на определенном узле окна Параметры, воспользуйтесь командой отменить внесенные изменения. Для полного восстановления настроек по умолчанию для определенного узла окна Параметры воспользуйтесь командой вернуться к значениям по умолчанию.

Рисунок 2.7– Откат настроек приложения

2.2 Автоматизация процесса

тестирования

Основными задачами при внедрении автоматизированного тестирования являются:

1) повышение качества тестирования. С помощью автоматизированного тестирования можно охватить больший набор тестов, и/или тесты, которые невозможно осуществить вручную. Кроме того,

68

автоматическое выполнение тестов позволяет свести к минимуму влияние человеческого фактора на результаты тестирования;

2) экономия времени, затрачиваемого на тестирование. На выполнение автоматизированных тестов обычно уходит значительно меньше времени, чем на выполнение аналогичных тестов вручную.

К преимуществам автоматизированного тестирования относится:

1) сокращение времени на исполнение тестов в сравнении с ручным тестированием;

2) возможность проведения тестов, которые нельзя провести [1]

вручную;

3) исключение человеческого фактора;

4)

возможность проводить тестирование вне рабочих часов тестировщика.

Необходимо также отметить и недостатки внедрения автоматизации тестирования;

1) автоматизация тестирования зачастую требует значительных трудозатрат;

2) требуется более квалифицированный персонал в [1]

сравнении с

ручным тестированием;

3) более сложный анализ результатов завершения автоматизированных скриптов;

4)

любое изменение в работе системы может потребовать трудозатратных изменений в автоматизированных тестах;

5) ошибка в реакции системы на один из тестов может привести к ошибочным результатам тестовой сессии всех последующих тестов;

6) риск возникновения ошибок в самом автоматизированном тесте;
7) в некоторых случаях, не все функциональные особенности системы можно покрыть автоматизированными тестами с помощью выбранного инструментария [38].
[1]

Недостатки автоматизированных скриптов:
69
1)

повторяемость – все написанные тесты всегда будут выполняться однообразно;
2) затраты на поддержку – несмотря на то, что в случае автоматизированных тестов они меньше, чем затраты на ручное тестирование того же функционала – они так же существуют. Чем чаще изменяется приложение, тем они выше;
3) затраты на разработку – это сложный процесс, так как фактически идет разработка приложения, которое тестирует другое приложение. В сложных автоматизированных тестах также есть фреймворки, утилиты, библиотеки и прочее. [3]

Как и основное приложение, это требует времени и сил
на тестирование и отладку;
4)

стоимость инструмента для автоматизации – в случае если используется лицензионное ПО, его стоимость может быть достаточно высока. Свободно распространяемые инструменты как правило отличаются более скромным функционалом и меньшим удобством работы;
5) пропуск мелких ошибок – автоматизированный скрипт может пропускать мелкие ошибки, на проверку которых он не запрограммирован. Это могут быть неточности в позиционировании окон, ошибки в надписях, которые не проверяются, ошибки элементов и форм, с которыми не осуществляется взаимодействие во время выполнения скрип��а.
[3]

Автоматизацию необходимо применять при наличии в системе или программном продукте следующих категорий:
1) труднодоступные места (

бэкенд процессы, логирование файлов, запись в БД);
2) часто используемой функциональности, риски от ошибок в которой достаточно высоки;
3) рутинные операции, такие как переборы данных, формы с большим количеством вводимых полей;
4) валидационные сообщения;
5) длинные end-to-end сценарии;
70
6) проверка данных, требующих точных математических расчетов.
А также, многое другое, в зависимости от требований к тестируемой системе и возможностей выбранного инструмента для тестирования.
Для более эффективного использования автоматизации тестирования необходимо разработать отдельные тест-кейсы, проверяющие:
1) базовые операции создания/чтения/изменения/удаления сущностей (так называемые CRUD операции - Create / Read / Update / Delete). Пример: создание, удаление, просмотр и изменение данных о пользователе;
2) типовые сценарии использования приложения, либо отдельные действия. Пример: пользователь заходит на почтовый сайт, листает письма, просматривает новые, пишет и отправляет письмо, выходит с сайта. Это endto- end сценарий, который проверяет совокупность действий;
3) интерфейсы, работы с файлами и другие моменты, неудобные для тестирования вручную. Пример: система создает некоторый .xml файл,

структуру [3]