ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК
КАФЕДРА ИНФОРМАЦИОННЫХ И РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

ИССЛЕДОВАНИЕ И РАЗРАБОТКА МЕТОДОВ ИНТЕЛЛЕКТУАЛЬНОГО ТЕСТИРОВАНИЯ ПРОГРАММНЫХ ПРОДУКТОВ

Магистерская диссертация
обучающегося по направлению подготовки
09.04.02 Информационные системы и технологии
очной формы обучения
группы 07001635
Медведевой Надежды Юрьевны

Научный руководитель
д.т.н., доцент
Польщиков К.А.
Рецензент
д.т.н., доцент
Черноморец А.А.

БЕЛГОРОД 2018

РЕФЕРАТ

Исследование и разработка методов интеллектуального тестирования программных продуктов – Медведева Надежда Юрьевна, магистерская диссертация, Белгород, Белгородский государственный национальный исследовательский университет (НИУ «БелГУ»), количество страниц 64, включая приложения 64, количество рисунков 21, количество таблиц 3, количество использованных источников 51.

КЛЮЧЕВЫЕ СЛОВА: информационная система, автоматизированное тестирование, TestComplete.

ОБЪЕКТ ИССЛЕДОВАНИЯ: процесс автоматического тестирования приложения АЦК-Финансы

ПРЕДМЕТ ИССЛЕДОВАНИЯ: программное приложение АЦК-Финансы

ЦЕЛЬ РАБОТЫ: повышение эффективности тестирования программного обеспечения IT-компании за счет разработки автоматизированной системы тестирования для продукта АЦК-Финансы, а также корректирования общего плана процесса разработки с учётом создания данной системы

ЗАДАЧИ ИССЛЕДОВАНИЯ: изучение теоретических данных по тестированию и автоматизированному тестированию; изучение основных фреймворков для написания автотестов и выбор оптимального; моделирование системы автоматизированного тестирования; программная реализация системы; оценка экономической эффективности данной разработки.

МЕТОДЫ ИССЛЕДОВАНИЯ: метод лексико-синтаксических шаблонов; статистические методы анализа текстов на естественном языке

ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ: Результатом выполнения магистерской диссертации является разработка автоматизированной системы тестирования программного обеспечения. Благодаря данной системе повысилась эффективность работы компании, а именно:

- сократилось время процесса тестирования;
- сократилось количество специалистов по тестированию;
- уменьшились затраты денежных средств на заработную плату.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Автоматизированное тестирование является неотъемлемой частью современного подхода к разработке программного обеспечения. Организация системы автоматического тестирования позволяет идентифицировать ошибки функциональности и дизайна приложения на всех этапах его создания [19].

Преимущества, которые предоставляет система [18]:

– При любом изменении, внесенном программистом в код, запускается тестовый сценарий, проверяющий корректную работу прежней и новой функциональности. Таким образом, разработчик всегда уверен в том, что его действия не повлекли за собой ошибок работы приложения. А если ошибка возникла, то отчёт о прохождении тестов позволит точно идентифицировать место и причину сбоя.

– Автоматизированное тестирование значительно сокращает время разработки ПО, так как заменяет процесс ручного тестирования. Кроме того, автономные тесты способны работать для разных форм и приложений. Таким образом, один раз написанный тест может служить инструментом для поиска ошибок многократно.

– Случайный подход генерации данных, используемых в тестах, позволяет найти непредвиденные ошибки. Так как очень часто машина способна сгенерировать тестовый пример, который ручной тестировщик мог не предусмотреть.

– История прохождения тестов на разных этапах разработки даёт ценную статистическую информацию: среднее количество ошибок при разработке одной формы, прохождении одного спринта (если речь идёт о компаниях, поддерживающих гибкие методологии разработки), за всё время создания продукта; время, потраченное на исправление ошибок; типы ошибок, их процентное соотношение. Такого рода информация может быть представлена заказчику, а также активно использоваться при дальнейшей

разработке, чтобы на её основании улучшить процесс, устраняя недостатки предыдущих релизов. Кроме того, информация может быть использована при прохождении проверок компании на соответствие тому или иному мировому стандарту.

Не смотря на все перечисленные преимущества автоматизированного тестирования, в большинстве компаний, занимающихся разработкой программного обеспечения, понятие ручного тестирования не превратилось в атавизм. Это происходит из-за того, что создание подобной системы является нетривиальной и очень специфической задачей. В зависимости от функциональности разрабатываемого программного обеспечения, языков и средств программирования, задача организации системы кардинально меняется, приобретая свои персональные проблемы.

Целью данной работы является повышение эффективности тестирования программного обеспечения IT-компании за счет разработки автоматизированной системы тестирования для продукта АЦК-Финансы, а также корректирования общего плана процесса разработки с учётом создания данной системы.

Задачи включают в себя:

– изучение теоретических данных по тестированию и автоматизированному тестированию;

– изучение основных фреймворков для написания автотестов и выбор оптимального;

– моделирование системы автоматизированного тестирования;

– программная реализация системы;

– оценка экономической эффективности данной разработки.

Объектом исследования данной работы является процесс автоматического тестирования приложения АЦК-Финансы.

Предмет исследования – программное приложение АЦК-Финансы, методы написания автотестов, модель работы системы автоматизированного тестирования.

Практическое значение создаваемой системы заключается в высвобождении человеческих ресурсов и увеличении экономического эффекта производимых тестовых операций.

Новизна магистерской диссертации заключается в разработке оригинального средства тестирования, которое позволит повысить эффективность работы компании за счет следующего:

– сокращения времени процесса тестирования;

– сокращения количества специалистов по тестированию;

– экономии денежных средств за счет уменьшения затрат на заработную плату.

Магистерская диссертаци состоит из 3 разделов:

– аналитическая часть – в данном разделе будет рассмотрен процесс автоматизированного тестирования, его место и роль в жизненном цикле программного обеспечения;

– построение системы – в данном разделе будут изучены основные системы написание автоматических тестовых сценариев и обоснован выбор среды, используемой в данной разработке, а также создана модель автоматизированной системы;

– проектная часть – в данном разделе будет представлен процесс разработки и анализа автоматического теста, также будет наглядно представлена экономическая эффективность от внедрения данной системы;

Магистерская диссертация написана на 64 страницах. Она включает в себя 21 рисунок, 3 таблицы и 4 формулы.

По окончании исследования планируется получение настраиваемой и функциональной системы автоматизированного тестирования.

1 Описание предметной области

1.1 Характеристика предприятия

Компания «Бюджетные и Финансовые Технологии» создана в 1997 году. На текущий момент занимает одно из лидирующих положений в сфере управленческих и информационных технологий для государственного и муниципального управления [44].

Входит в группу компаний IBS – лидера рынка информационных технологий и консалтинга России.

Тесно сотрудничает с органами федеральной власти – Администрацией Президента РФ, Советом Федерации, Министерством финансов РФ, Федеральным казначейством, Федеральной Антимонопольной службой и многими другими.

Компания БФТ входит в состав [44]:

– экспертной группы при Координационной комиссии по созданию и развитию государственной интегрированной информационной системы управления общественными финансами «Электронный бюджет»;

– рабочей группы Министерства финансов Российской Федерации по вопросам совершенствования государственного (муниципального) контроля.

Направление деятельности Компании – обеспечение возможности эффективно и своевременно решать задачи государственного и муниципального управления и оказания государственных и муниципальных услуг в каждом поселении, каждом муниципальном образовании, каждом регионе России.

Продуктовая линейка Компании охватывает ключевые сферы государственного и муниципального управления [44]:

– автоматизированная система управления государственными финансами (АСУ ГФ);

– отраслевые решения;

– услуги и общественный контроль;

– управление имущественными и земельными отношениями;

– электронный документооборот;

– автоматизация деятельности МФЦ;

– методическое сопровождение деятельности ОГВ и ОМС (Консалтинг);

– защита информации.

Компанией БФТ разработан ряд решений для коммерческих организаций и банков.

Программные продукты БФТ разрабатываются с учетом бюджетной реформы, всех требований законодательства РФ и интегрированы друг с другом.

Проекты в области консалтинга реализованы в 16 регионах и около 100 муниципальных образованиях, а также в Министерстве спорта РФ, Министерстве финансов РФ, Министерстве образования и науки РФ. Благодаря этим проектам решаются следующие задачи:

– разработка стратегий социально-экономического развития и иных документов стратегического планирования;

– переход на программный бюджет;

– формирование программной классификации расходов бюджета;

– совершенствование межбюджетных отношений;

– разработка и экспертиза государственных (муниципальных) Программ по повышению эффективности бюджетных расходов и иным актуальным направлениям;

– повышение качества управления государственными (муниципальными) учреждениями в рамках реализации Федерального закона;

– разработка отраслевых систем оплаты труда. Переход на эффективный контракт;

– правовой консалтинг: консультационное сопровождение создания и реорганизации государственных (муниципальных) учреждений и предприятий;

– консультационное сопровождение перехода к контрактной системе;

– обучение клиентов, проведение семинаров и вебинаров.

1.2 Место тестирования в разработке программного обеспечения

В первую очередь стоит отметить, что процесс тестирования ПО тесно связан непосредственно с процессом разработки. Жизненный цикл разработки состоит из следующих этапов [34]:

– анализ требований;

– дизайн;

– разработка;

– тестирование и дебаггинг;

– эксплуатация и поддержка.

Как показано в списке, мы должны провести тестирование на четвертом шаге жизненного цикла. Но обычно в случае, если нашей главной целью является получить высококачественное ПО и минимизировать затраты на исправление багов, мы можем проводить тестирование уже на стадии анализа требований. Чем раньше вы приступите к тестам, тем лучших результатов добьетесь.

Детально рассмотрим какие преимущества может принести проведение тестирования на каждом этапе процесса разработки, начиная с самого первого.

Первый этап жизненного цикла разработки - это анализ требований. Требования к конечному продукту обычно формулируются заказчиком или менеджером проекта. Эти требования могут быть как функциональными, так и нефункциональными. Они формируются в процессе общения с заказчиком или анализа стандартов и нормативной документации [29].

Именно на этом этапы жизненного цикла необходимо начать проводить тесты ПО. Если имеющиеся требования не были протестированы, но были использованы на этапе дизайна и разработки. Только после того, как разработка закончена, требования и сам продукт направляются в отдел тестирования. Как было сказано ранее, в процессе тестирования мы проверяем, соответствует ли текущее поведение продукта заявленным требованиям. А это значит, что отдел тестирования может обнаружить ошибки не только в самом продукте, но также и в документации. В этом случае исправление ошибок обойдется гораздо дороже в сравнении с подходом, который предусматривает включение тестов в самые ранние этапы жизненного цикла ПО, такие как фаза анализа требований. Тщательным образом проанализировав требования, вы можете собрать информацию, которая поможет вам оптимизировать процесс работы над проектом с самых первых дней. Как правило, тестирование требований происходит на этапе анализа требований. Эта задача подразумевает ряд тестов, основанных на таких характеристиках как завершенность, согласованность, недвусмысленность и т.д. Главная задача такого подхода — убедиться в том, что требования заказчика были правильно интерпретированы и остаются корректными, понятными и последовательными. Важно отметить, что ясная и точная документация помогает выбрать правильные цели для процесса тестирования.

Следующим этапом жизненного цикла разработки ПО является процесс дизайна. Как и тестирование требований на стадии анализа требований, этот этап подразумевает проверку уже созданных прототипов и мокапов на предмет их корректности и соответствия ожиданиям заказчика.

Более того, проверка удобства в использовании также должна быть проведена на этом этапе. Также следует начать создание тестовой документации для данного проекта. Эта задача включает в себя подготовку плана тестирования, тест-кейсов, юзкейсов, а также другой документации по требованию заказчика. Процесс тестирования ПО на этом этапе обеспечивает способность проникновения в суть продукта и понимание ее соответствия требованиям. Важным является точное понимание задач, стоящих перед отделом тестировапния на протяжении всего жизненного цикла разработки.

В течении этапа разработки важно провести модульное, интеграционное и системное тестирование. В самом начале этого шага разработки проводится модульное тестирование. Этот процесс представляет собой проверку отдельного модуля системы или функционала. Интеграционное тестирование проводится после того, как несколько модулей объединены вместе как отдельная часть приложения. В дальнейшем в процессе разработки все больше и больше модулей объединяются воедино. После того, как разработка закончена, наступает время подготовки к системному тестированию. Эта стадия жизненного цикла разработки ПО подразумевает общий тест системы на предмет интеграции ее компонентов. Это значит, что в случае, если система состоит из различных модулей, мы должны проверить, насколько хорошо или насколько плохо каждый из них работает внутри системы. Более того, на этом этапе важно произвести тестирование пользовательского интерфейса.

Четвертый этап жизненного цикла – это процесс тестирования и дебаггинга.

На этом шагу вы должны провести тесты независимо от того, проводились ли они на предыдущих этапах. Должны быть проведены полное функциональное тестирование и тестирование пользовательских интерфейсов, а все обнаруженные дефекты должны быть задокументированы в системе баг-трекинга. Помимо этого, применяется регрессионное тестирование. После завершения дебаггинга предоставляется оценка общего

качества продукта. После завершения последнего теста процесс тестирования ПО считается законченным [42].

Пятый этап - эксплуатация и поддержка.

Даже после достижения стадии релиза продукта, остается необходимость в тестировании, проводимом на этапе эксплуатации и поддержки. Разные пользователи могут работать в абсолютно разных окружениях. Поэтому всегда возможно, что новые ошибки, которые не были выявлены ранее, дадут о себе знать. Более того, пользователи могут использовать ПО изначально непредвиденным способом. Это, в свою очередь, может вызвать некоторые непредвиденные проблемы. В таком случае потребуется вмешательство отдела тестирования.

Очевидно, что процесс управления тестированием ПО затрагивает все этапы жизненного цикла разработки. Он подразумевает сравнение действительного состояния продукта и того состояния, которое было запланировано и задокументировано в плане тестирования продукта. Процесс тестирования, анализа и мониторинга помогает спланировать и изменить последующие задачи наилучшим путем.

Жизненный цикл тестирования ПО является процессом, которого нельзя избежать. Он непрерывен, продолжителен и требует наличия команды специалистов по тестированию, достаточно опытной для того, чтобы произвести полный цикл тестирования. Эта неотъемлемая часть современного процесса разработки ПО помогает заказчику, команде разработчиков, а также конечному пользователю получить продукт высокого качества.

1.3 Виды тестирования

Все виды работ проводимых тестировщиком можно условно разделить на два: функциональное и нефункциональное тестирование [38].

Функциональное тестирование — тестирование программного обеспечения, главная цель которого это проверка реализуемости функциональных требований приложения, т.е. способность приложения в заданных критериях решать возложенные на него (на приложение) задачи. Требования включают в себя:

- защищенность;
- соответствие стандартам;
- способность к взаимодействию;
- функциональная пригодность;
- точность.

Не функциональное тестирование ПО — в первую очередь проверка на соответствие не функциональным требованиям:

- удобство - в основном производиться оценка удобства для пользователей;
- маштабируемость - проверяется как вертикальная так и горизонтальная маштабируемость тестируемого приложения;
- производительность - способность работы приложения при различных нагрузках;
- безопасность - защита пользовательских данных, защита данных приложения, стойкость на взлом;
- портируемость - совместимость и переносимость приложения для и под различные окружения, платформы и т.д.;
- надежность - поведение системы при различных непредвиденных ситуациях, способность обработки нестандартных действий пользователя.

В свою очередь, эти виды тестирования ПО имеют по множеству разнообразных разбиений по особенностям тестирования. Так же тестирование может еще подразделяться на уровни тестирования, которые в той или иной мере могут пресекаться между собой.

Еще существует более детальное разбиение по целям, хронологии, знанию системы, сценариям и т.д.

Чтобы было немного проще понять все градации тестирования приведем рисунок 1.

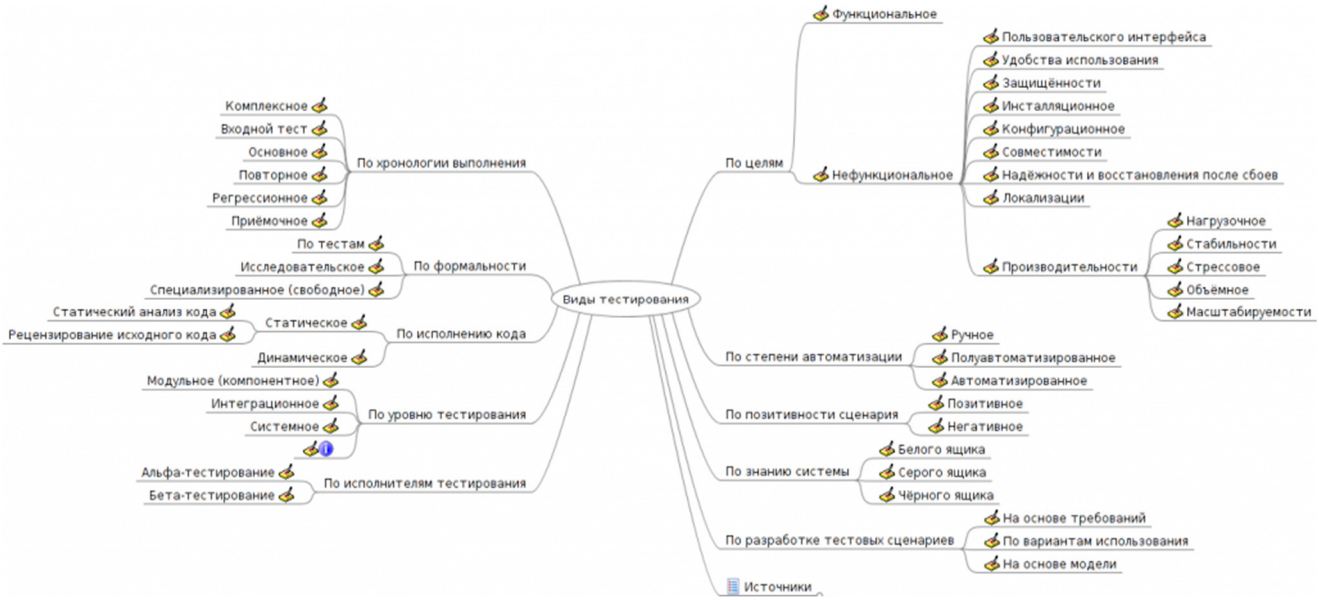

Рисунок 1 - Виды тестирования ПО

По объектам, которые подвергаются тестированию [49]:

- функциональное тестирование;
- тестирование локализации и интернационализации;
- тестирование взаимодействия;
- конфигурационное тестирование;
- тестирование производительности;
- тестирование стабильности;
- стресс-тестирование;
- нагрузочное тестирование;
- юзабилити-тестирование;
- тестирование документации;
- тестирование интерфейса пользователя;
- тестирование безопасности.

По степени знания тестируемой системы:

- тестирование чёрного ящика;
- тестирование белого ящика;

- тестирование серого ящика.

По степени изолированности части компонентов тестируемого ПО:

- модульное тестирование;
- интеграционное тестирование;
- системное тестирование.

По степени глубины тестирования:

- тестирование критического пути;
- расширенное тестирование.

По времени проведения тестирования:

- альфа-тестирование;
- дымовое тестирование (smoke testing) ;
- тестирование новой функции (new feature testing) ;
- регрессионное тестирование;
- приёмочное тестирование;
- бета-тестирование.

По признакам позитивности сценариев

- негативное тестирование;
- позитивное тестирование.

По критериям запуска программы или программного кода:

- статическое тестирование;
- динамическое тестирование.

По степени подготовки к тестированию тетировщиком:

- интуитивное тестирование (ad hoc testing) ;
- тестирование по документации (формальное тестирование).

По степени автоматизации процесса тестирования:

- автоматизированное тестирование;
- ручное тестирование.

Автоматизация - это комплекс мероприятий, направленных на повышение производительности труда человека посредством замены части этого труда работой машины.

Автоматизация тестирования - это использование программного обеспечения для выполнения или поддержки тестирования, тест-дизайна, выполнения тестов, анализа результатов выполнения тестов и т.д. Автоматизация тестирования - это не только выполнение автоматических тестов, это написание и использование всевозможных скриптов для анализа результатов, подготовки тестовых данных, парсинга документов и т.д., т.е. автоматизация всех рутинных и повторяющихся задач для облегчения процесса тестирования [50].

По большому счету тестирование можно разделить на 3 вида: ручное, автоматизированное и частично автоматизированное.

Ручное тестирование. Данный вид означает, что все тестирование проводится руками, без использования скриптов и написание автотестов. Т.е. все тестовые данные составляются и забиваются руками; прогон тестов так же осуществляется руками; ну и конечно происходит ручной анализ прогона тестов и ручная фиксация багов.

Автоматизированное тестирование. Здесь уже идет максимальная автоматизация процесса тестирования. Т.е. тестовые данные генерируются автоматически; прогон тестов выполняется автоматически; ну и происходит автоматический анализ прогона тестов и автоматическая фиксация багов.

Частично автоматизированное тестирование. В основном используется в тех случаях, когда либо нет необходимости все автоматизировать, либо нет возможности, денег или времени на автоматизацию. В данном виде тестирования используется полуавтоматическая генерация тестовых данных; тесты выполняются в автоматическом или полуавтоматическом режиме; происходит ручной анализ прогона тестов и ручная фиксация багов [45].

Как уже было сказано, автоматизация служит для облегчения работы тестировщика и улучшения качества тестирования, а значит и продукта. Основные цели, для которых служит автоматизация, изображены на схеме (рисунок 2).

Рисунок 2 – Цели автоматизации

Т.е. автоматизация покрывает весь спектр работ по тестированию программного продукта.

Ручное и автоматизированное тестирования сегодня играют существенную роль в любой технологической компании. Будь то мобильное или web-приложение или сайт, проверка кода крайне важна. Правильное планирование, когда и какое тестирование использовать, помогает сохранять время и деньги [30].

Обе методики тестирования имеют свои преимущества и недостатки, их мы рассмотрим ниже.

Прежде чем начать выбирать вид тестирования, необходимо составить подробный план жизненного цикла приложения или web-сайта. Планируете ли вы:

- Вносить много изменений?
- Добавлять новый функционал?
- Полностью обновлять приложение или web-сайт?

Это очень важно, ведь вышеперечисленные и многие другие факторы увеличивают жизненный цикл вашего продукта.

Ручное тестирование может занимать много времени, зато в краткосрочной перспективе сэкономит в разы больше денег. Его стоимость зависит только от тестировщика, а не инструментов для автоматизации.

Ручное тестирование можно рассматривать как взаимодействие профессионального тестировщика и софта с целью поиска багов. Таким образом, во время ручного тестирования можно получать фидбек, что невозможно при автоматизированной проверке. Иными словами, взаимодействуя с приложением напрямую, тестировщик может сравнивать ожидаемый результат с реальным и оставлять рекомендации [21].

Если у вас есть QA-команда, ручное тестирование не будет проблемой.

К плюсам ручного тестирования можно отнести:

- пользовательский фидбек - весь отчёт тестировщика может быть рассмотрен как обратная связь от потенциального пользователя;
- UI-фидбек - в наше время пользовательский интерфейс играет огромную роль, и поэтому полностью протестировать его можно только вручную;
- дешевизна - в краткосрочной перспективе ручное тестирование дешевле, чем инструменты автоматизированной проверки;
- тестирование в реальном времени - незначительные изменения могут быть исследованы сразу, без написания кода и его исполнения;
- возможность исследовательского тестирования - его целью является проверка разнообразных возможностей приложения. Важно, что используются не заранее составленные тест-кейсы, а придуманные сценарии.

Минусами ручного тестирования являются:

- человеческий фактор - хотя UI и может быть протестирован только вручную, люди часто склонны к неэффективности. Некоторые ошибки могут остаться незамеченными;
- трудоемкость повторного использование - провести серию стандартных автоматических тестов проще, чем протестировать проект вручную после внесения даже небольших изменений;
- невозможность нагрузочного тестирования - нельзя смоделировать большое количество пользователей вручную.

Автоматизированное тестирование — это написание кода. С его помощью ожидаемые сценарии сравниваются с тем, что получает пользователь, указываются расхождения. Автоматизированное тестирование играет важную роль в тяжёлых приложениях с большим количеством функций [23].

Плюсы автоматизированного тестирования это:

- возможность нагрузочного тестирования - можно достаточно быстро смоделировать большое количество пользователей;
- экономия времени - ручное тестирование больших приложений — долгий и трудоёмкий процесс, в то время как сценарии пишутся лишь один раз;
- возможность повторного использования - тестовый сценарий, написанный один раз, может быть использован и в будущем при очередном обновлении проекта.

К минусам автоматизированного тестирования относятся:

- дороговизна - инструменты автоматизированного тестирования, а также обучение их использованию стоят недёшево, поэтому нужно тщательно оценивать бюджет;
- UI-тестирование - автоматизированное тестирование не может в полной мере покрыть требования к пользовательскому интерфейсу;
- отсутствие «человеческого взгляда» - возможно существование ошибок, которые заметит только человек.

Далее следует рассмотреть такое понятие, как тестовая система. Зачастую под тестовой системой понимают набор автоматических тестов, что абсолютно не верно. Тестовая система представляет связку из тестовой логики, реализации тестовой логики, запуск тестов и системы отчетов. Более наглядно это выглядит на рисунке 3.

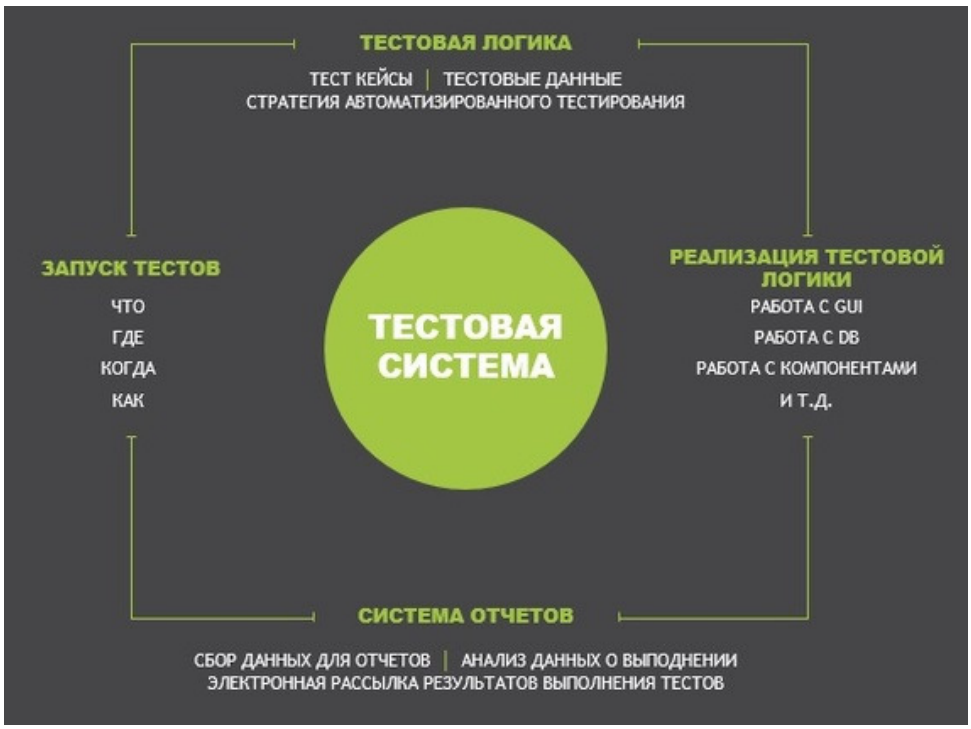

Рисунок 3 – Тестовая система

Автоматические тесты являются отдельным программным продуктом. А это значит, что для этого нужны соответствующие специалисты, они так же могут содержать багги, и требуется достаточно времени для их имплементации [48].

Из плюсов можно отметить следующие:

– увеличение скорости тестирования без ущерба для результата;

– возможность выполнять тесты 24 часа 7 дней в неделю;

– уменьшение объема ручных, рутинных, постоянно повторяющихся операций в процессе тестирования;

– снижение стоимости итерации тестирования;

– автоматическое формирование отчетов о тестировании;

– набор тестов ограничивается в первую очередь производительностью системы, а не доступным резервом времени тестировщиков.

Помимо плюсов, у автоматизации есть целый ряд минусов, которых необходимо учесть при выборе вида тестирования. Итак, минусы:

– автоматизация тестирования - это дорого;

– плохо спроектированные или реализованные автоматические тесты ведут к нестабильным результатам и ложным срабатываниям;

– требуются люди с высокой квалификацией в области тестирования и знанием языков программирования и технологий;

– автоматические тесты "не смотрят" по сторонам;

– требуется время на поддержку;

– баги в автотестах ведут к пропуску багов в продакшен;

– высокая стоимость платных инструментов.

Как и у каждого вида деятельности, в автоматизации тестирования есть свои подводные камни:

– не все тест-кейсы можно\нужно автоматизировать;

– для автоматизации тестирования необходимо разработать соответствующее программное средство. Следовательно, тут проявляются все проблемы, связанные с разработкой;

– автоматические тесты требуют хорошей проработки архитектуры тестовой системы, в противном случае у вас к ручному тестированию добавится работа по поддержке автоматических тестов;

– на старте автоматизации нельзя предугадать\предусмотреть все риски и проблемы, с которыми вы столкнетесь при автоматизации.

Существуют несколько причин, чтобы начать использовать автоматизацию. Для наглядности изобразим все на рисунке 4.

Рисунок 4 – Причины начала автоматизации

Точного ответа на вопрос "Автоматизировать или не автоматизировать?" - нет. Ответ всегда зависит от целей, задач проекта и команды.

1.4 Понятие тестового сценария

Тестирование представляет собой процесс проверки того, насколько программное обеспечение соответствует требованиям, заявленным заказчиком. Он осуществляется в специальных, искусственно создаваемых ситуациях посредством наблюдения за работой программного обеспечения. Такого рода искусственно построенные ситуации называют тестовыми или просто тестами.

Разработка тестов происходит на основе проверяемых требований и критерия полноты тестирования. Разработанные тесты формируются в тест-кейс (набор тестов) и выполняются на ПО, которое нужно протестировать. После прогона всех тестов анализируется результат, в результате чего можно выявить ошибки в программе [37].

Процесс тестирования схематично показан на рисунке 5.

Рисунок 5 - Процесс тестирования

Тест-кейс (тестовый случай) - это минимальный элемент тестирования (одна проверка), содержащий в себе описание конкретных действий, условий и параметров, которые направлены на проверку какой-либо функциональности. Набор тест-кейсов называется тестовым набором (test suite) [33].

Тест-кейсы позволяют тестировщикам провести проверку продукта без полного ознакомления с документацией. При условии, что 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.

Тест-кейс должен включать в себя:

– Уникальный идентификатор тест-кейса. Этот идентификатор необходим для удобства организации и навигации по тестовым наборам.

– Название. В названии должна отражаться основная идея тест-кейса, цель данной проверки.

– Предусловия. Список шагов, не имеющих прямого отношения к проверяемому функционалу, которые необходимо выполнить до начала теста. Например, для тест-кейса «Заказ товара» предусловием может быть шаг «авторизоваться на сайте», если на данном сайте заказать товар может только авторизованный пользователь.

– Шаги. Описание последовательности действий, которая должна привести к ожидаемому результату.

– Ожидаемый результат. Поведение системы, предусмотренное требованиями. Один тест-кейс проверяет одну конкретную функцию, поэтому ожидаемый результат может быть только один.

– Статус кейса. 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-найдена ошибка.

c) Выполнение теста блокировано, если после одного из шагов продолжение теста невозможно. В этом случае так же, найдена ошибка.

– История редактирования. Дает возможность узнать, кем и когда был изменен тест-кейс. Это удобно, поскольку позволяет более эффективно редактировать тест-кейсы.

Тест-кейс должен отвечать следующим требованиям [21]:

– Для измерения покрытия требований, 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.

– Тест является хорошим в случае, когда он может обеспечить высокую вероятность обнаружения ошибки. Показать, что в программе полностью отсутствуют ошибки невозможно, поэтому процесс тестирования должен быть направлен на выявление прежде не найденных ошибок.

– Четкие, однозначные формулировки шагов. Описание шагов для прохождения тест-кейса должно содержать всю необходимую информацию, но при этом тест-кейс не должен быть слишком детализирован. 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.

– Отсутствие зависимостей тест-кейсов. Если тесты связанны между собой, становится проблематичным изменение, дополнение или удаление конкретного тест-кейса, появляется необходимость изменять связанные с ним тесты. Более того, взаимосвязанные тесты обладают конкретный сценарием от перехода одного теста к другому. Это приведет к

тому что не все сценарии перехода от одного теста к другому будут протестированы и появляется вероятность пропустить баг.

– Ожидаемый результат необходимо прогнозировать заранее и прописывать его в тест-кейсе. Если ожидаемый результат не определить заранее, может возникнуть ситуация, когда тестировщик видит то, что он хочет увидеть. При заранее определенном результате тестировщику необходимо только сравнить ожидаемый результат с фактическим.

– Необходимо уделять внимание не только тестам, которые проверяют правильные данные, но и тем тестам, которые проверяют работу программы при неправильных, непредусмотренных данных. Большое количество ошибок связано именно с теми действиями пользователя, которые не предусмотрены программой.

– Также необходимо проверять, не делает ли программа то, чего не должна. Нужно производить проверку на нежелательные побочные эффекты.

1.5 Выводы по первому разделу

Раздел «Описание предметной области» является теоретическим обоснованием необходимости создания разработки автоматизированной системы. В данной главе было изучено понятие автоматизированного тестирования, его основные термины и виды.

Также был проведен анализ деятельности специалиста по тестированию и изучено понятие тестовый сценарий.

Так как разработка системы осуществляется для ООО «БФТ», мы изучили общую характеристику компании и основные направления ее деятельности.

2 Построение системы интеллектуального тестирования

2.1 Выбор инструмента автоматизации

Существует множество методов для написания автотестов. Рассмотрим основные из них и выберем тот который подойдет нам.

Автономный тест – это часть кода (обычно метод), которая вызывает другую часть кода и затем проверяет правильность некоторых предположений. Если предположения не подтверждаются, считается, что автономный тест завершился неудачно. Автономной единицей (unit) является метод или функция [5].

NUnit-тестирование – это такой подход к тестированию, когда берется минимально возможный кусок кода, изолируется от другого кода и проводятся тесты, помогающие понять, что эта часть кода работает так, как ожидается. В более узком смысле unit-тестирование подразумевает что, во-первых, кусочек кода, который мы тестируем – это метод класса, во-вторых, все тесты, которые мы проводим автоматизировано, и, в-третьих, время, которое требуется на выполнение тестов достаточно маленькое, так что мы можем запускать их так часто как мы этого хотим.

NUnit - это модульная платформа для всех Net-языков. NUnit - это программное обеспечение с открытым исходным кодом, а NUnit 3 выпускается под лицензией MIT . В более ранних версиях использовалась лицензия NUnit. Обе эти лицензии позволяют использовать NUnit в бесплатных и коммерческих приложениях и библиотеках без ограничений [8].

Модульный тест – это код с определенной структурой, который проверяет поведение одного класса или функции (т.е. использует внутренние интерфейсы приложения). Модульный тест написан на том же языке

программирования, что и тестируемое приложение и пишется, как правило, самим разработчиком.

Модульные тесты необходимы для:

– повышения качества продукта:

a) предотвращение ошибок и их быстрое выявление;

b) изолированная проверка работоспособности внутренних модулей для быстрой локализации дефектов;

– понимание системы:

a) тесты как спецификация системы;

b) тесты как документация;

– снижение рисков:

a) быстрое выявление побочных эффектов изменений в коде;

b) безопасная работа с унаследованным кодом, который покрыт модульными тестами.

Все семейство фреймворков для написания модульных тестов объединилось под определением XUnit.

XUnit – это семейство структур автоматизации тестирования реализующих общие правила и предназначенных для автоматизации созданных вручную тестов. С другой стороны, xUnit – это общие правила, которые реализуют эти фреймворки [2].

К общим принципам xUnit-тестирования относится:

– тест – это тестовый метод (метод класса, процедура, функция), реализующий четырехфазный тест;

– объединение тестовых методов в классы в коде;

– использование утверждений для проверки поведения системы;

– объединение тестов в тестовые наборы на этапе выполнения;

– обнаружение или явное перечисление тестов;

– различные варианты запуска тестов;

– отчеты о результатах тестирования.

WatiN означает «Тестирование веб-приложений в NET» и получен из WaTiR, который представляет собой «Тестирование веб-приложений в Ruby» [6].

Структура WatiN обеспечивает тестирование веб-приложений через Internet Explorer, но также поддерживает Firefox. Он поддерживает версии Internet Explorer 6, 7, 8 и Internet Explorer 9 и версии Firefox 2.x и 3.x.

Он имеет библиотеку с открытым исходным кодом и известен очень простым для чтения синтаксисом.

WatiN разработан в C# и нацелен на то, чтобы предоставить простой способ автоматизации тестов.

WatiN взаимодействует с Internet Explorer, используя интерфейс COM (Component Object Model). Взаимодействие Firefox выполняется с использованием расширения JSSh (серверной оболочки JavaScript) для Mozilla Firefox, присутствующего в файле WatiN.Core.dll .

JSSH - это подключаемый модуль, который предоставляет сервер telnet, позволяющий командам автоматизации использовать Firefox. Когда WatiN загружается, он содержит архив с файлом WatiN.Core.dll .

Работать с WatiN очень легко. WatiN позволяет открывать экземпляры Internet Explorer, взаимодействуя с элементами в форме. С WatiN вы можете получать и устанавливать значения из элементов в форме, и вы можете запускать события любого из элементов.

Сегодня нетрудно проверить код. На рынке для тестирования кода доступно множество инструментов. Но если мы говорим о веб-приложениях, то появляется важный слой, который мы никогда не игнорируем, это слой пользовательского интерфейса.

Одна из проблем, возникающих при написании автоматизированных тестов для элементов управления в пользовательском интерфейсе - это проверки функциональности. Иногда бывает так, что наш тест терпит неудачу, потому что автоматическое тестирование не может найти элемент управления в приложении.

Несмотря на то, что разработчик добавил его, хорошим способом является использование классов WatiN и автоматизация.

Предварительные требования для WatiN включают:

– структура Visual Studio для написания тестовых скриптов в C #;

– NUnit или Gallio Test Runner для выполнения тестовых сценариев.

Используя Watin, мы можем выполнить следующее тестирование в веб-приложении ASP.NET:

– тестирование GUI;

– функциональное тестирование;

– регрессионное тестирование.

Ниже перечислены особенности WatiN:

– автоматизация всех элементов HTML;

– поиск элементов по нескольким атрибутам;

– тестирование веб-сайта Ajax;

– синтаксис WatiN более объектно-ориентирован и интуитивно понятен для разработчиков;

– поддержка создания скриншотов веб-страниц;

– простота интеграции с вашим любимым инструментом тестирования;

– может использоваться с любым языком NET;

– поскольку он имеет открытый исходный код, вы можете загружать и добавлять новые функции самостоятельно.

Ниже приведены ограничения WatiN:

– обработка Ajax и автоматизация Silverlight;

– одним из недостатков структуры WatiN является то, что коды статуса HTTP не отображаются, что в настоящее время является ограничением автоматизации COM в Internet Explorer;

– если вы используете IE7 и выше, он включает в себя функцию под названием «Защищенный режим». Этот режим вызывает проблемы, когда

WatiN пытается получить доступ к веб-странице. Таким образом, разумно отключить его через настройки безопасности в Интернете для браузера. Это может привести к проблемам безопасности.

TOSCA Testsuite - это программный инструмент для автоматического функционального и регрессионного тестирования программного обеспечения. В дополнение к функциям автоматизации тестирования TOSCA включает интегрированное управление тестированием, графический интерфейс пользователя (GUI), интерфейс командной строки (CLI) и интерфейс прикладного программирования (API). TOSCA Testsuite разработан австрийской компанией программного обеспечения TRICENTIS Technology & Consulting GmbH, базирующейся в Вене [3].

Основные характеристики TOSCA:

- Динамическое рулевое управление: концепция, стоящая за TOSCA Commander, - это подход, основанный на модели, чтобы сделать «весь» тест, а не только входные данные, динамическими. Тестовые примеры создаются путем перетаскивания модулей и ввода значений и действий проверки. Предполагается, что динамизация теста позволит основать бизнес-описание ручных и автоматизированных тестовых примеров, чтобы тестовые примеры могли быть спроектированы, указаны, автоматизированы и поддерживаться нетехническими пользователями (МСП).
- Основные характеристики Tosca Testsuite включают в себя создание динамических синтетических тестовых данных, высоко автоматизированное динамическое управление бизнес-процессами и унифицированное управление и выполнение ручных и автоматизированных, а также графических и не GUI-тестов.
- Кроме того, тестовые примеры могут быть взвешены в соответствии с их важностью при бесперебойной работе бизнес-процесса. Таким образом, TOSCA предоставляет подробную отчетность, которая показывает влияние существующих технических слабых мест на выполнение

требований. Например, Fecher использует тестовый инструмент в новых разработках и проектах миграции приложений и баз данных.

Электронная автоматизация тестирования поддерживается для следующих технологий:

- языки программирования и фреймворки: Delphi, .NET, включая WPF, Java Swing / SWT / AWT, Visual Basic;
- среда разработки приложений: Gupta, PowerBuilder;
- веб-браузеры: проводник, Firefox;
- хост-приложения в 3270, 5250;
- ключевые прикладные программы: SAP, Siebel;
- однопозиционные прикладные программы: Outlook, Excel;
- аппаратное обеспечение и протоколы: исполнение USB, Flash, SOAP (WebServices), ODBC.

Selenium - это инструмент для автоматизированного управления браузерами. Наиболее популярной областью применения Selenium является автоматизация тестирования веб-приложений. Однако при помощи Selenium можно автоматизировать любые другие рутинные действия, выполняемые через браузер [7].

Разработка Selenium поддерживается производителями популярных браузеров. Они адаптируют браузеры для более тесной интеграции с Selenium, а иногда даже реализуют встроенную поддержку Selenium в браузере. Selenium является центральным компонентом целого ряда других инструментов и фреймворков автоматизации.

Selenium поддерживает десктопные и мобильные браузеры. Selenium позволяет разрабатывать сценарии автоматизации практически на любом языке программирования. С помощью Selenium можно организовывать распределённые стенды, состоящие из сотен машин с разными операционными системами и браузерами, и даже выполнять сценарии в облаках.

Selenium состоит из нескольких частей.

Selenium IDE - расширение браузера Firefox, которое позволяет записывать и воспроизводить действия пользователя в браузере. Он создает:

- небольшой сценарий для быстрого автоматизированного воспроизведения бага;
- вспомогательный скрипт для выполнения отдельных рутинных действий при ручном тестировании.

Selenium WebDriver - набор библиотек для различных языков программирования, позволяющих управлять браузером из программы, написанной на этом языке программирования.

Он необходим, если требуется разработать:

- надежный фреймворк автоматизации, способный работать с любым браузером;
- большой тестовый набор, включающий тесты с достаточно сложной логикой поведения и проверок.

Selenium Server - может принимать команды с удалённой машины, где работает сценарий автоматизации, и исполнять их в браузере. Несколько серверов Selenium могут образовывать распределённую сеть, которая называется Selenium Grid, что позволяет легко масштабировать стенд автоматизации.

Он необходим для:

- запуска тестов удалённо на разных машинах с разными операционными системами и браузерами;
- организации тестового стенда для выполнения большого количества тестов.

TestComplete – это автоматизированное средство тестирования, позволяющее создать тесты для Windows-приложений, web-серверов и web-страниц.

TestComplete используется для автоматизации различных типов тестирования программного обеспечения для .NET, Java, Visual C++, Visual Basic, Delphi, C++ Builder, web-страниц и других приложений. TestComplete

помогает тестировщикам разрабатывать свои тест-кейсы в различных скриптовых языках: JavaScript, Python, VBScript, Delphi Script и JavaScript.

В TestComplete существует специальный механизм, облегчающий создание скриптов для тестирования приложений. Суть механизма заключается в следующем: тестировщик выполняет необходимые действия, которые автоматически записываются в файл. Данный файл, который доступен для просмотра и редактирования, при запуске повторяет все те операции, которые были предварительно выполнены. Важным обстоятельством является тот факт, что записанные тесты могут быть изменены позднее [4].

TestComplete обладает широкими возможностями в том, что касается автоматизации тестирования:

- тестирование ключевых слов: используя встроенный редактор, пользователи могут разработать специальный фреймворк;
- скрипт-тест: тестировщики могут писать тестовые скрипты с нуля или модифицировать записанные прежде во встроенном редакторе;
- запись теста и воспроизведение: предоставляет базовый механизм записи и воспроизведения для создания теста. По необходимости записанные тест-кейсы можно видоизменять;
- интеграция в баг-трекеры : интегрируется с различным софтом для отслеживания дефектов (Jira, Bugzilla). Может использоваться для изменения и создания элементов в баг-трекере, для чего имеется специальный шаблон;
- тестирование данных: удобное извлечение данных из CSV-файлов, таблиц баз данных, страниц Excel и прочего;
- Test Visualizer (визуализация тестирования): делает скриншоты во время выполнения теста, таким образом можно сравнить с ожидаемым результатом то, что на экране.

Существует ряд инструментов для выполнения качественного автоматизированного тестирования разных веб-продуктов. TestComplete является одним из таких серий продуктов.

TestComplete – это самый популярный фреймворк из данной серии. Кроме того, он является самым простым в использовании. Относительно часто компании по тестированию программного обеспечения используют именно данный инструмент.

TestComplete – это набор программных средств для автоматизированного тестирования, который позволяет создавать и запускать тесты для любых десктопных, веб и мобильных приложений, а так же управлять ими. Выполняя тестирование веб-приложений с применением TestComplete, специалисты смогут ощутить ряд преимуществ, которых нет у других фреймворков. А именно:

- решение TestComplete располагает встроенным редактором тестов, который включает в себя ключевые слова и операции;
- с помощью TestComplete, можно создавать мощные и гибкие тестовые скрипты как новичкам, так и профессионалам;
- возможно тестирование как Windows-приложений, так и Web-приложений;
- в TestComplete возможно гибко распределять нагрузки на несколько машин. Если необходимо, можно добавить еще один тестовый компьютер и выполнять больший объем тестов;
- в программе реализована возможность синхронизировать тесты, запущенных на разных рабочих станциях;
- поддержка разнообразных типов тестирования, например: модульное тестирование, Web-тестирование, функциональное (GUI) тестирование, регрессионное тестирование, ключевое тестирование и тестирование мобильных приложений;
- TestComplete поддерживает 32-разрядные и 64-разрядные приложения;

- поддержка: JIRA Software, Software Planner и других программ для отслеживания;
- включен открытый API;
- легкая расширяемость с помощью плагинов;
- обширная база документации и примеров;
- решение совместимо с современными операционными системами: Windows 7, Vista, XP, 2000, 8, 8.1, 10; Windows Server 2003, 2008,2012.

2.2 Проектирование автоматизированной системы тестирования

Функциональная модель системы предназначена для изучения особенностей работы системы и её назначения во взаимосвязи с внутренними и внешними элементами [35]. Функциональная модель системы, разработанная в среде AllFusion Process Modeler, приведена на рисунке 6.

Входными параметрами системы являются: исходные настройки системы, новая сборка, скрипт автотеста; выходными: результат выполнения автотеста.

Механизмами на данной схеме представлен отдел автотестирования. А управляющими стрелками являются: действующая сборка программы и требования заказчика.

На рисунке 7 представлена декомпозиция диаграммы, показывающая детальную работу автоматизированного теста, состоящую из трех этапов:

- чтение настроек;
- проверка необходимости установки сборки;
- проверка наличия тестов для выполнения.

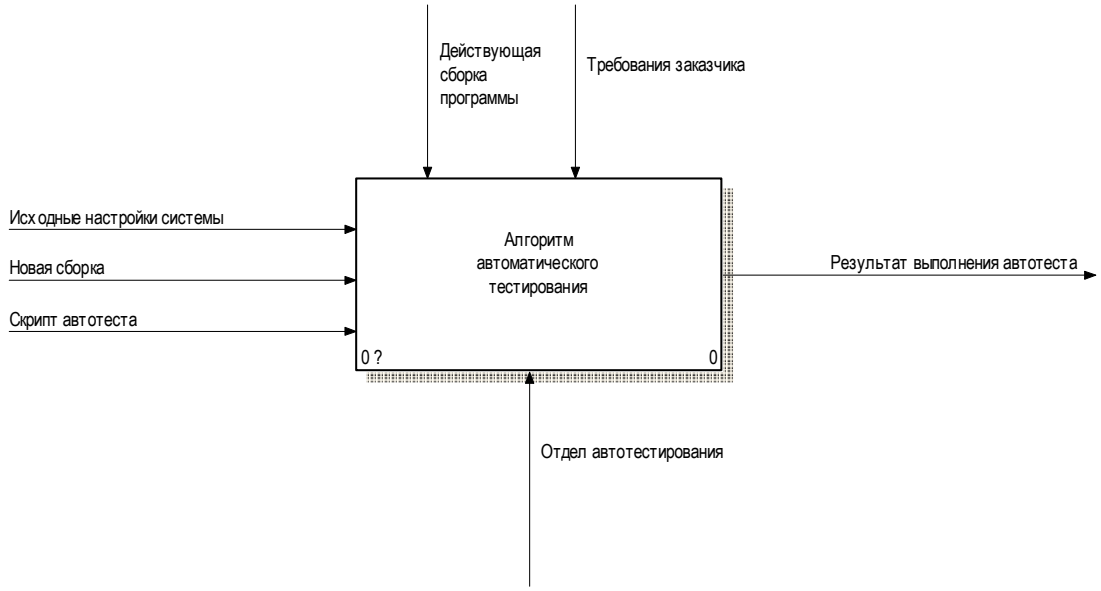

Рисунок 6 – Контекстная диаграмма

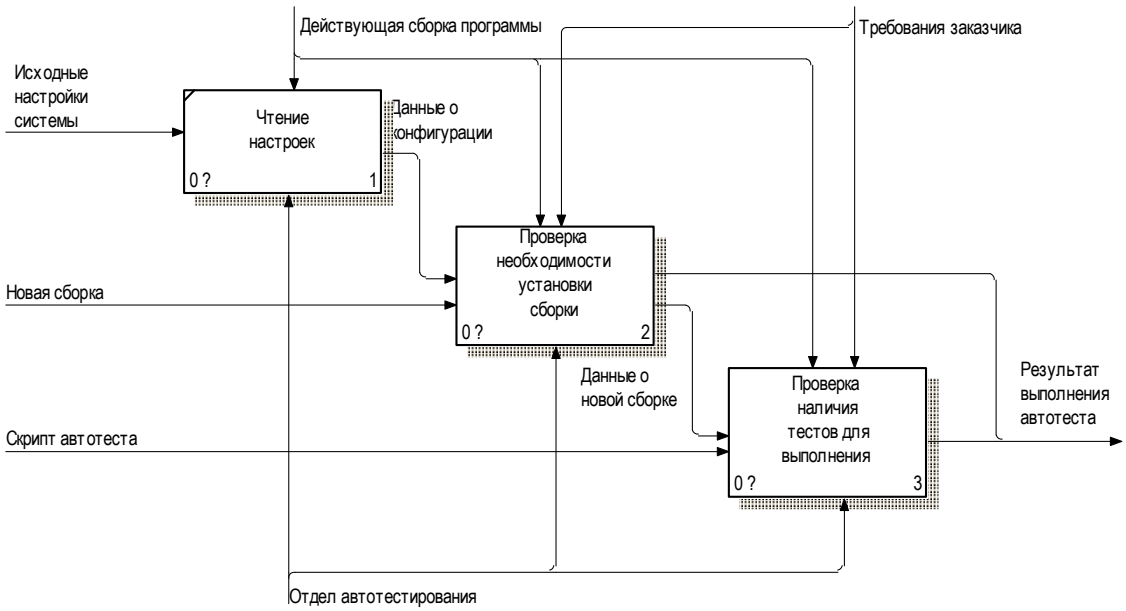

Рисунок 7 - Алгоритм автоматизированного тестирования

Процесс «Проверка необходимости установки сборки» также декомпозируется (рисунок 8):

- проверка необходимости установки новой сборки;
- проверка наличия сборки;
- удаление старой сборки;
- установка новой сборки;
- проверка корректности установки;

– проверка количества допустимых установок;

– переход к автотесту.

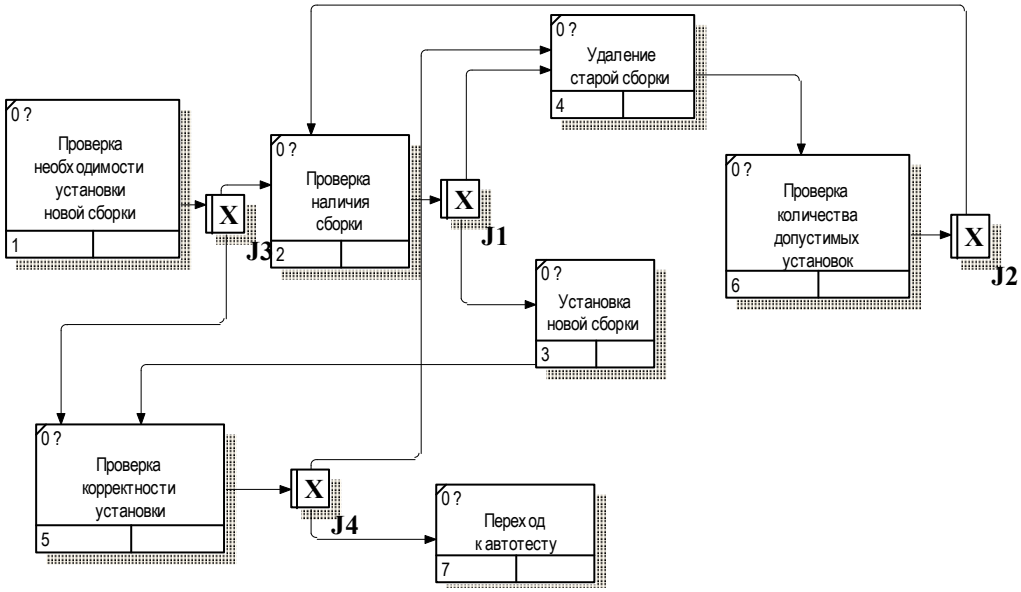

Рисунок 8 – Проверка необходимости установки сборки

Процесс «Проверка наличия тестов для выполнения» также декомпозируется (рисунок 9):

– проверка наличия теста для выполнения;

– выполнение очередного теста;

– сохранение результата;

– сообщение о выполнении тестов.

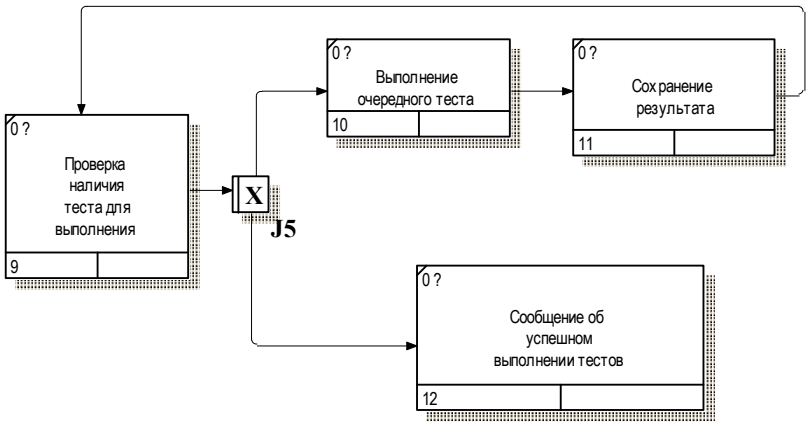

Рисунок 9 – Проверка наличия тестов для выполнения

2.3 Выводы по второму разделу

Во втором разделе было рассмотрено программное обеспечение, которое используется для создания автоматизированных тестов, и приведено обоснование выбора программного средства TestComplete.

Также была спроектирована модель работы автоматизированной системы тестирования.

3 Программная реализация системы

3.1 Создание структуры автотеста

Все данные, необходимые для автоматизированного тестирования, для удобства использования помещаются в одну папку autotests, которая содержит следующие подкаталоги [46]:

– ActionBuilder – содержит приложение, с помощью которого составляются xml с тестами;

– apps – содержит приложения, используемые во время выполнения автотестов;

– git-tester – содержит ключи для работы пользователя на виртуальных машинах, на которых прогоняются автотесты;

– LoadTests – содержит нагрузочные тесты;

– TC_Runner – содержит приложение для запуска автотеста;

– test-selenium – проект для поддержки автотестов веб-клиента с использованием библиотеки selenium;

– test-selenium-siar – проект автотестов СИАР;

– test-vaadin – проект для поддержки автотестов веб-клиента, реализованного на платформе qdp;

– TestFramework – содержит папки с автотестами по направлениям и папку Scripts с модулями для поддержки работы автотестов;

– Actions – содержит xml-файты с тестовыми сценариями, которые разделены по следующим типам (подкаталогам): dictionary – тесты справочников, documents – тесты документов, reports – тесты отчетов, setup – тесты предзаполнения;

– Templates – содержит шаблоны-описатели форм справочников и документов;

– reports_etalons – содержит эталонные файлы для тестов отчетов;

– files содержит файлы с данными для тестов. Данные вынесены в отдельные файлы, а в самих тестах используются ссылки на эти данные. Такой способ хранения выбран по причине неоднократного использования данных, чтобы в случае изменения данных можно было исправлять только в одном месте. Также в папке files хранятся xml, копируемые в сборку при установке проекта, сертификаты для подписи.

Во время выполнения установки проекта создается папка data, в которую копируются данные для установки проекта, и происходит его установка. Папка data содержит следующие подкаталоги [34]:

– build – папка, в которую при установке копируются данные из сборки – архивы клиента, web-клиента, сервера, отчетов, томката и т.д.

– db – папка, в которую при установке копируется бэкап базы или заархивированная база. В эту папку распаковывается и в ней хранится база firebird;

– deploy – папка, в которую устанавливается сервер, клиент и web-клиент;

– reports – папка, в которую во время тестов сохраняются файлы отчетов.

3.2 Настройка ActionBuilder

Xml с тестами составляются с помощью ActionBuilder. Для его настройки необходимо выполнить следующие шаги [27]:

– Запускаем ActionBuilder.exe.

– Переходим в меню настроек Settings\Options.

– В поле Current version указываем папку с версией тестов.

– В поле Data path указываем путь до тестов.

– Сохраняем настройки (рисунок 10).

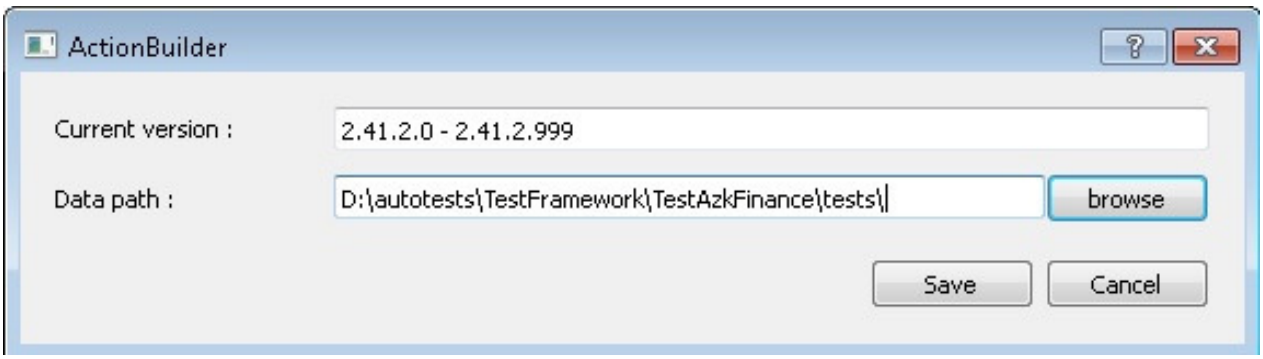

Рисунок 10 – Настройка пути и версии тестов

– На форме ActionBuilder в поле Test description пишем название теста.

– В выпадающем меню выбираем тип объекта, который нам нужен.

– В списке ниже выбираем нужный объект, разворачиваем его дерево и видим список контролов объекта. Разворачивая каждый контрол, можно увидетьсписок действий для него. Два поля внизу формы группы Data используются для заполнения параметров действий. После выбора действий над полями для параметров подсказки, какими данными их нужно заполнить.

– После выбора контрола действия для него и заполнения параметров нужно нажать кнопку Add. В правой части формы отобразится сформированной действие. С помощью кнопки Del действие можно удалить (рисунок 11).

– Таким способом добавляем новые действия. После того, как создадим все необходимые действия, сохраняем сформированный xml-файл.

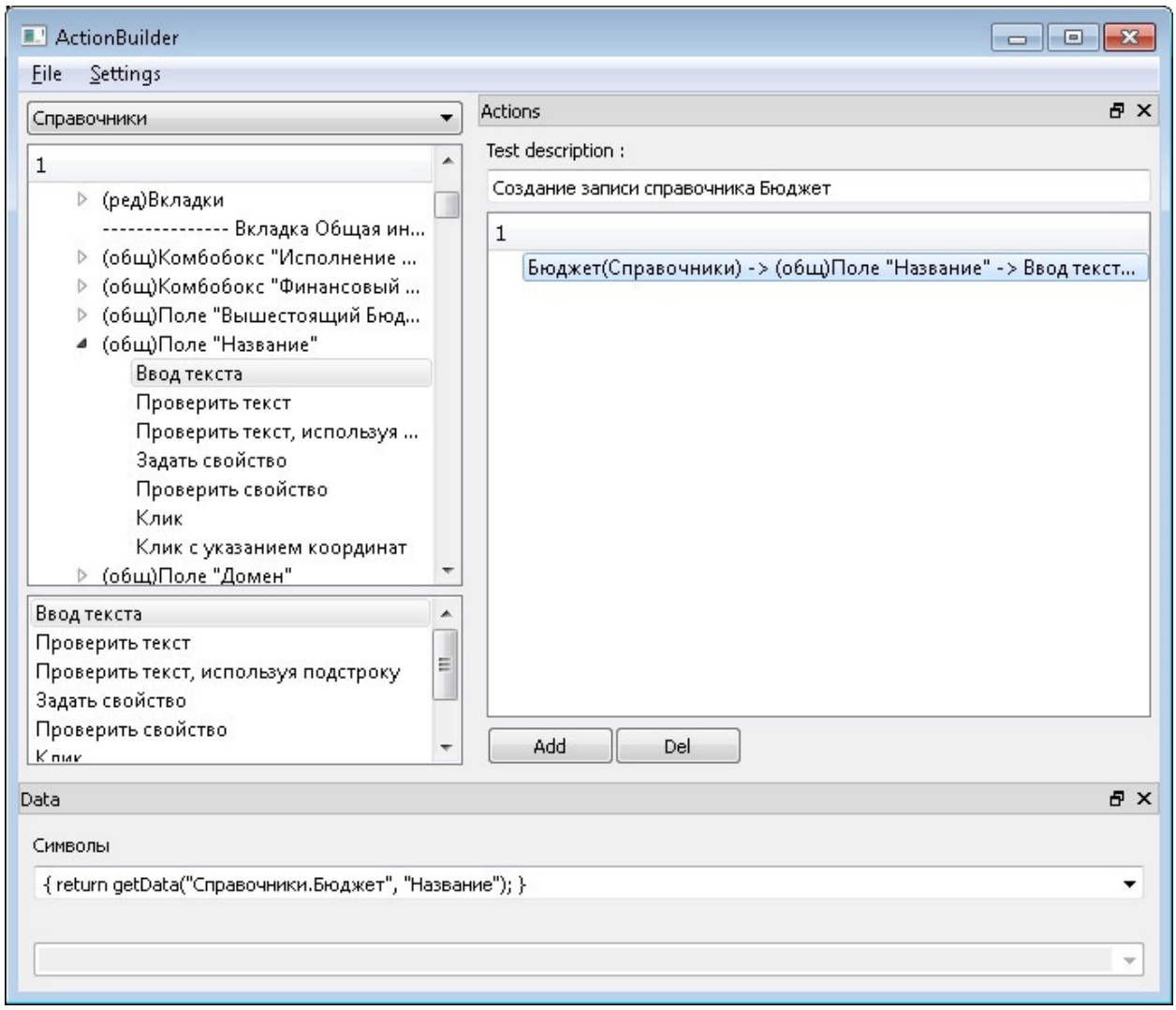

Рисунок 11 – Настройка автотеста в ActionBuilder

3.3 Создание структуры проекта

В Project Explorer отображается структура проекта автотестов. В Scripts мы видим подключенные к проекту модули. В папке common аходятся общие для разных проектов файлы. В папке core находятся ядерные модули. Они так же общие для различных проектов. В подпапке base расположены базовые модули, такие как модули для работы со строками, логами, почтой, загрузкой данных из xml-файлов, инсталляторы и т.д [26].

Стоит обратить внимание на следующие модули:

– core_base_defines – в нем описаны переменные для автотестов, общие для разных проектов: пути к серверу, клиенту, настройки для разных БД и т.д.;

– core_base_fr, core_base_fr4k – модули ля формирования шаблонов форм.

В подкаталоге control расположены модули для работы с контролами delphi-клиента.

В подкаталоге player расположены модули для поиска контролов (core_player_finder) и воспроизведения тестов (core_player) delphi-клиента.

В подкаталоге processes расположены модули для работы с различными процессами, например, сервером приложения, томкатом, delphi-клиентом, cmd-файлами, WinRar и т.д.

В папке project находятся проектные модули (для каждого проекта свои). В модуле fin_defines описаны проектные переменные, например, различные пути, используемые для конкретного проекта. Модуль fin_installer переопределяет некоторые методы основного инсталлера, таким образом, позволяя установить данные проекта, которые используются только в нем, к примеру, различные xml, проливаемые при установке.

В папке tests расположены модули с функциями, запускающими выполнение автотестов:

– TestReference – модуль с тестами справочников;

– tests_setup – модуль с тестами наполнения базы.

В TestedApps видим приложение, которое используется во время выполнения автотестов.

На вкладке Test items проекта видим дерево тестов, которое включает в себя установку проекта, проверку дерева навигации, тесты справочников и документов и т.д. (рисунок 12)

Для запуска «Проверка билда АЦК-Финансы» указываем параметры такие, как изображено на рисунке 13.

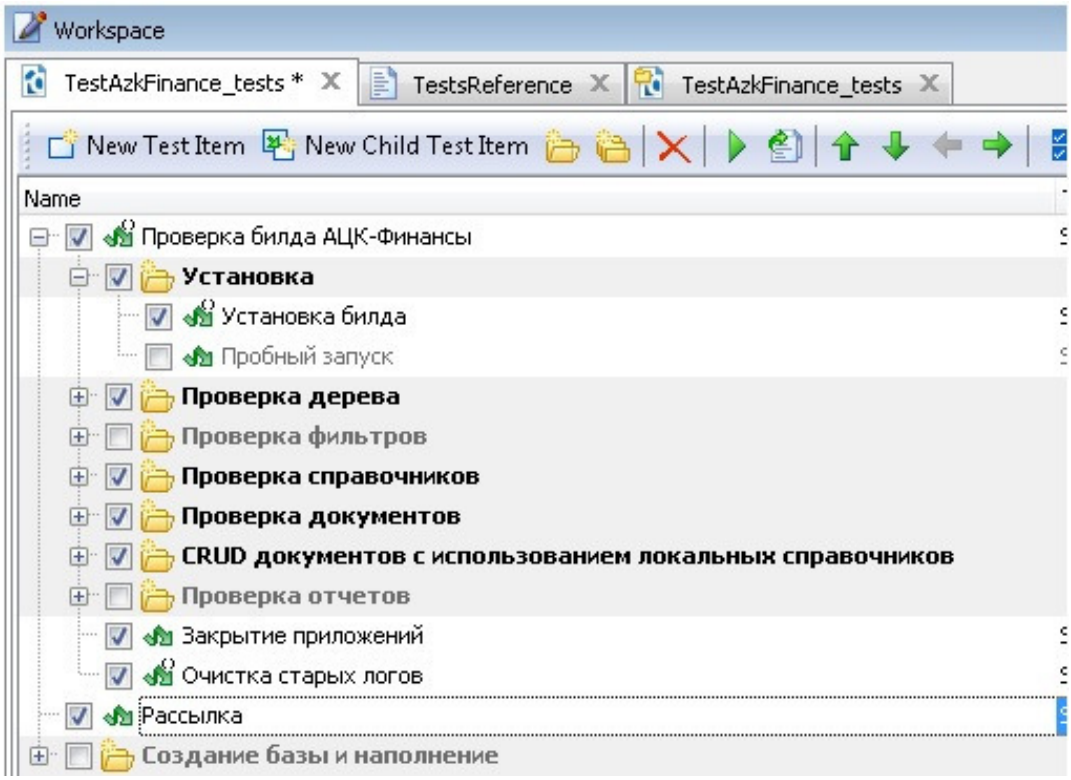

Рисунок 12 – Дерево тестов

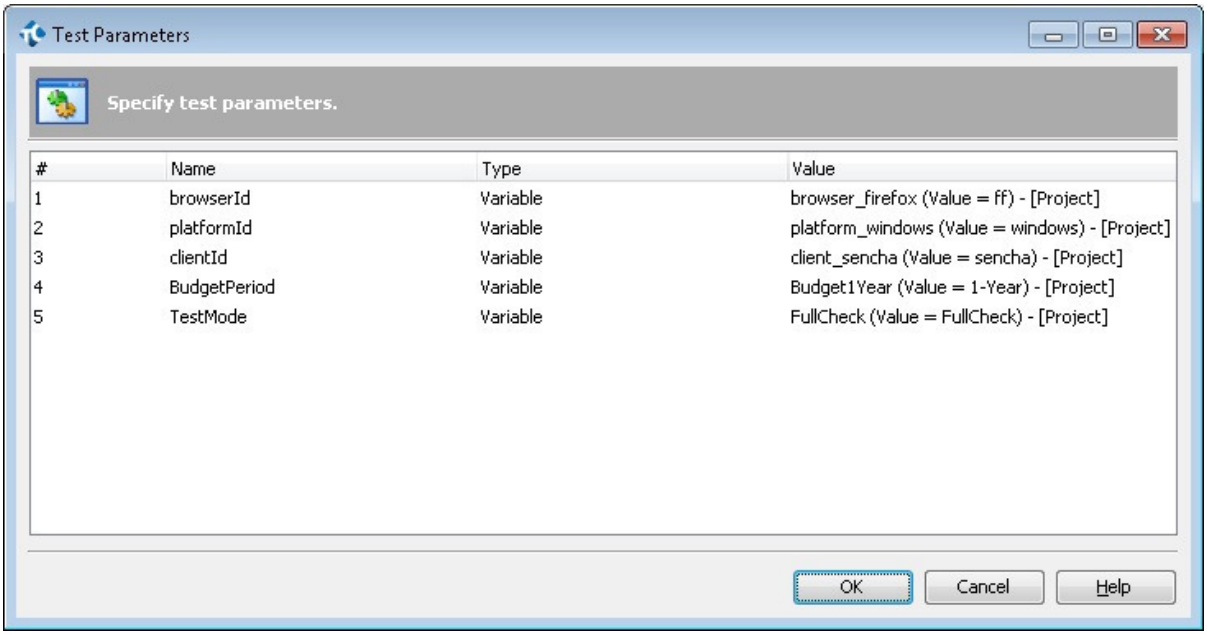

Рисунок 13 – Параметры для теста

Данные параметры заполняются проектными переменными, значения которых можно посмотреть на вкладке Variables проекта (рисунок 14).

Persistent Variables

Name	Type	Default Value	Local Value	Description
name	String	АЦК-Финансы	АЦК-Финансы	Название проекта
db_firebird	String	firebird	firebird	Для работы под firebird
db_oracle	String	oracle	oracle	Для работы под oracle
platform_windows	String	windows	windows	Для работы под windows
platform_redhat	String	redhat	redhat	Для работы под RHEL
browser_firefox	String	ff	ff	Для тестирования под Mozilla FireFox
browser_ie	String	ie	ie	Для тестирования под Internet Explorer
browser_chrome	String	cr	cr	Для тестирования под Google Chrome
browser_opera	String	opera	opera	Для тестирования под Opera
client_delphi	String	delphi	delphi	Для тестирования Delphi-клиента
client_sencha	String	sencha	sencha	Для тестирования Sencha-клиента
client_swt	String	swt	swt	Для тестирования SWT-клиента
mode_create	String	create	create	Тесты с созданием БД с нуля
mode_update	String	update	update	Тесты с использованием ранее созданной с нуля БД
mode_preset	String	preset	preset	Тесты с использованием предзаполненной БД
mode_stand	String	stand	stand	Установка Тестового стенда
mode_stand_update	String	stand_update	stand_update	Установка Демонстрационного стенда
mode_product	String	product	product	Установка Продукт-стенда
Budget1Year	String	1-Year	1-Year	Запуск на однолетнем бюджете(документы)
Budget3Year	String	3-Year	3-Year	Запуск на многолетнем бюджете(документы)
FullCheck	String	FullCheck	FullCheck	Полный прогон списка тестов
MinCheck	String	MinCheck	MinCheck	Запуск тестов по укороченному сценарию
db_postgre	String	postgre	postgre	Для работы под postgre

Рисунок 14 – Вкладка Variables

Перед запуском установки проекта необходимо указать следующие параметры (рисунок 15).

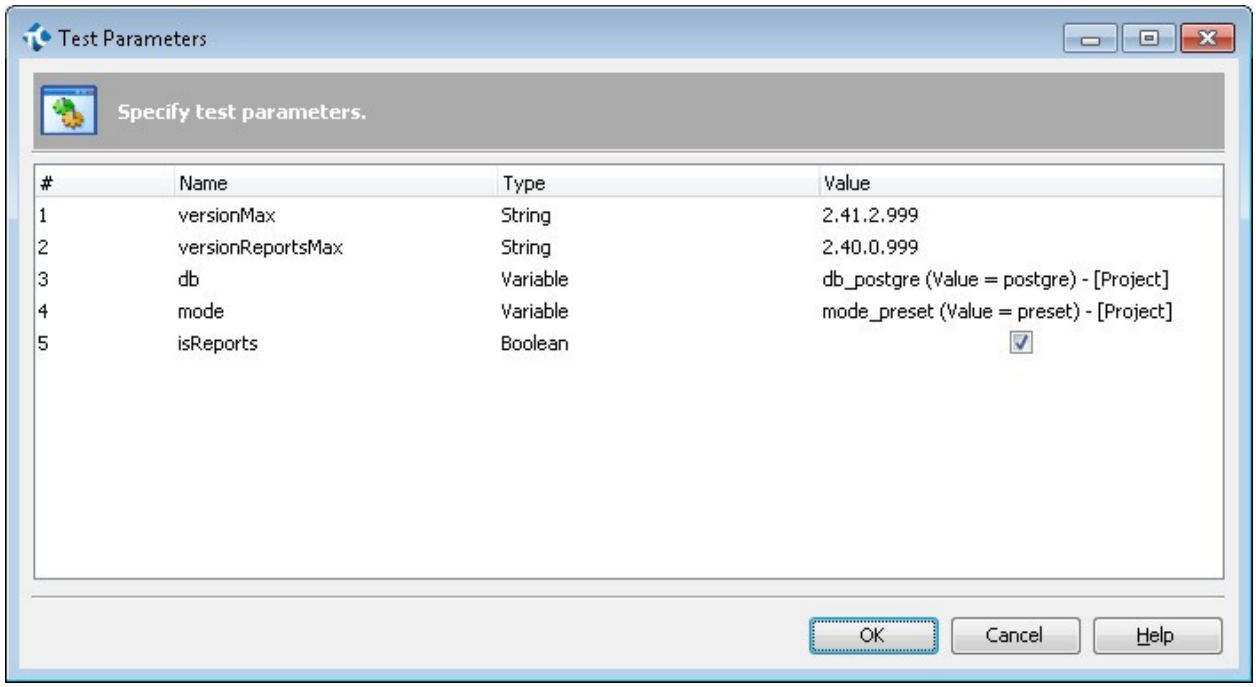

Рисунок 15 – Параметры для запуска

3.4Создание автотеста

XML (Extensible Markup Language) - это новый производный язык разметки документов, позволяющий структурировать информацию разного типа, используя для этого произвольный набор инструкций [42].

Сегодня XML может использоваться в любых приложениях, которым нужна структурированная информация - от сложных геоинформационных систем, с гигантскими объемами передаваемой информации до обычных "однокомпьютерных" программ, использующих этот язык для описания служебной информации. При внимательном взгляде на окружающий нас информационный мир можно выделить множество задач, связанных с созданием и обработкой структурированной информации, для решения которых может использоваться XML.

Разработку автотеста необходимо начать с создания xml-файл для справочника «сайты». Это можно сделать с помощью встроенного дополнения в TestComlete, которое называется Object Spy (рисунок 16). Данное дополнение позволяет задавать адрес, свойства и методы контрола. Если выбрать способ захвата «Point and fix», при наведении на любой контрол будет появляться красная рамка, и после захвате контрола можно выбрать нужный метод.

Рисунок 16 – Окно программы Object Spy

В данной XML все имена созданы без имени, поэтому их необходимо отредактировать в текстовом редакторе.

Далее перемещаем xml-файл в папку с действующей сборкой программы (рисунок 17).

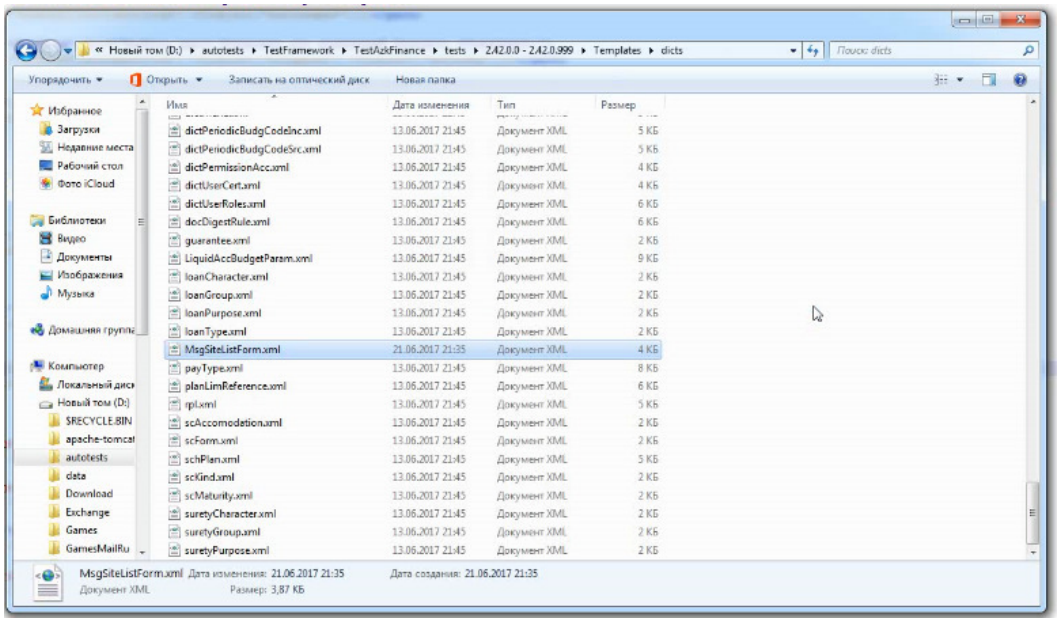

Рисунок 17 – Папка со сборкой программы

После этого в Action Builder записываем последовательность действий, которую будет выполнять системы в процессе тестирования ПО (рисунок 18).

Далее добавляем автотест в дерево тестов нашего проекта на вкладке Test Items (рисунок 19).

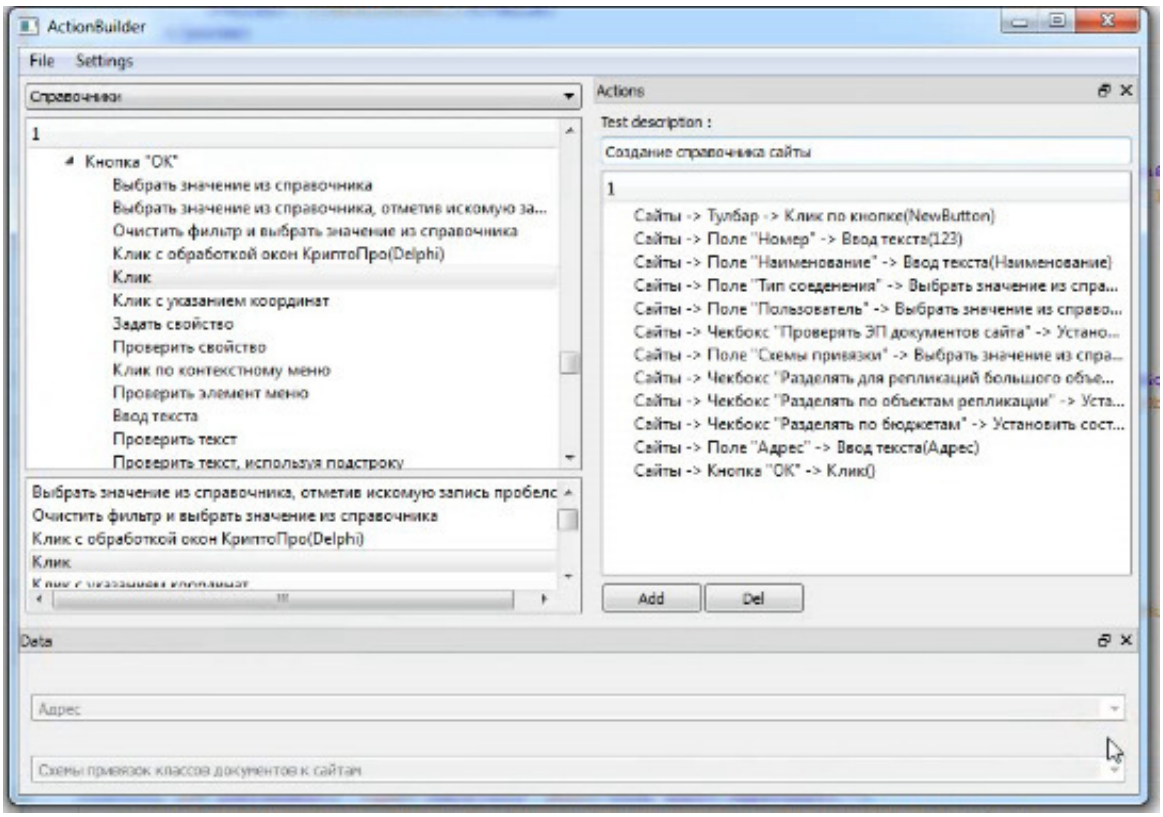

Рисунок 18 – Окно Action Builder

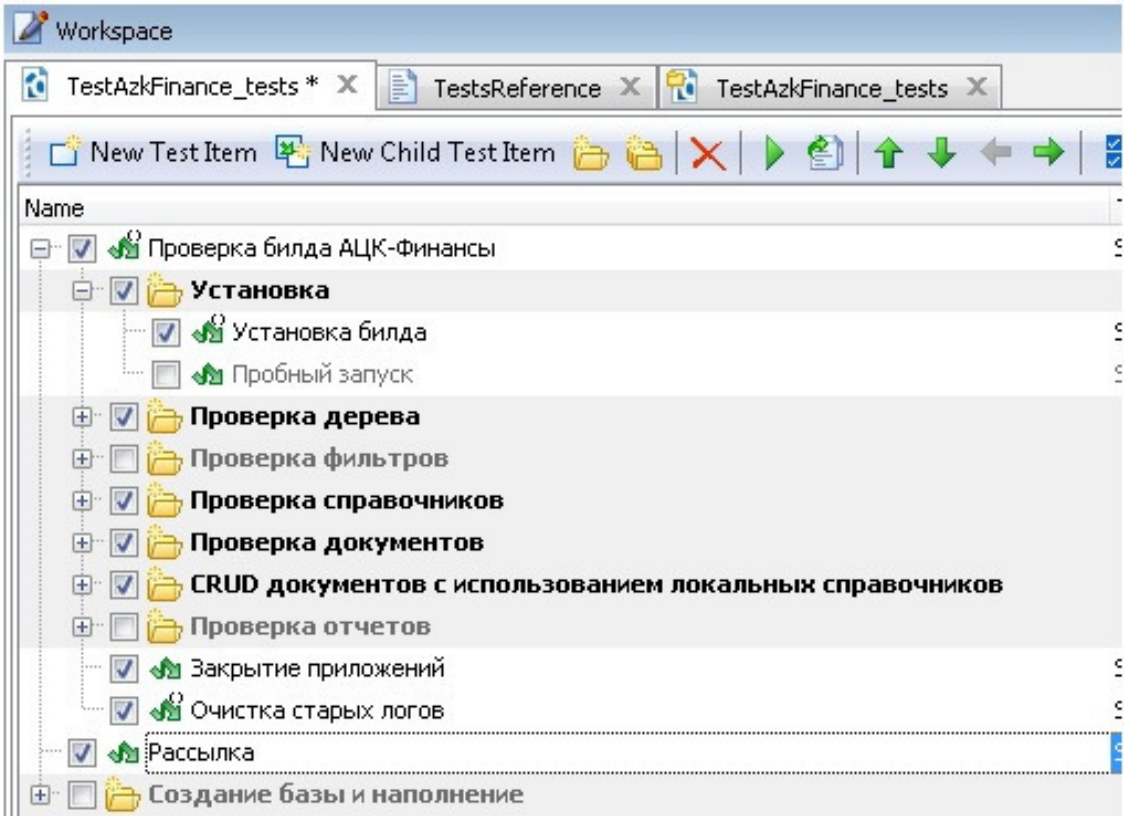

Рисунок 19 – Дерево тестов

3.5 Тестирование программы

После создания структуры автотеста и размещения его в дереве тестов, можно приступать к тестированию системы. Для этого запускаем необходимый тест с помощью кнопки «RUN». Это вызывает запуск программы и поочередное выполнение всех действий, который прописаны в автотесте. При успешном выполнении автотеста в TestComplete появляется сообщение, изображенное на рисунке 20.

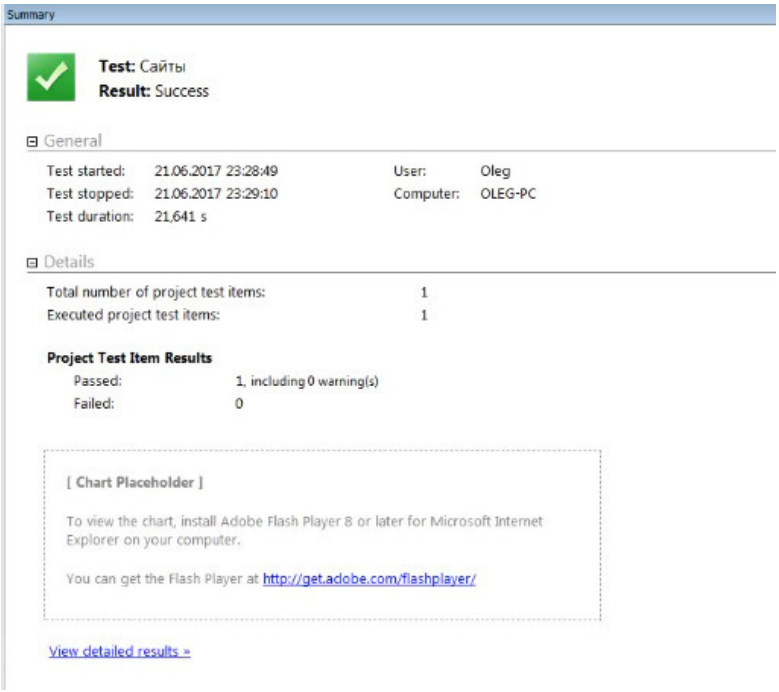

Рисунок 20 – Успешное выполнение теста

При раскрытии дерева результата, можно увидеть подробный отчет о выполнении автотеста и прохождении каждого шага тестирования программы (рисунок 21).

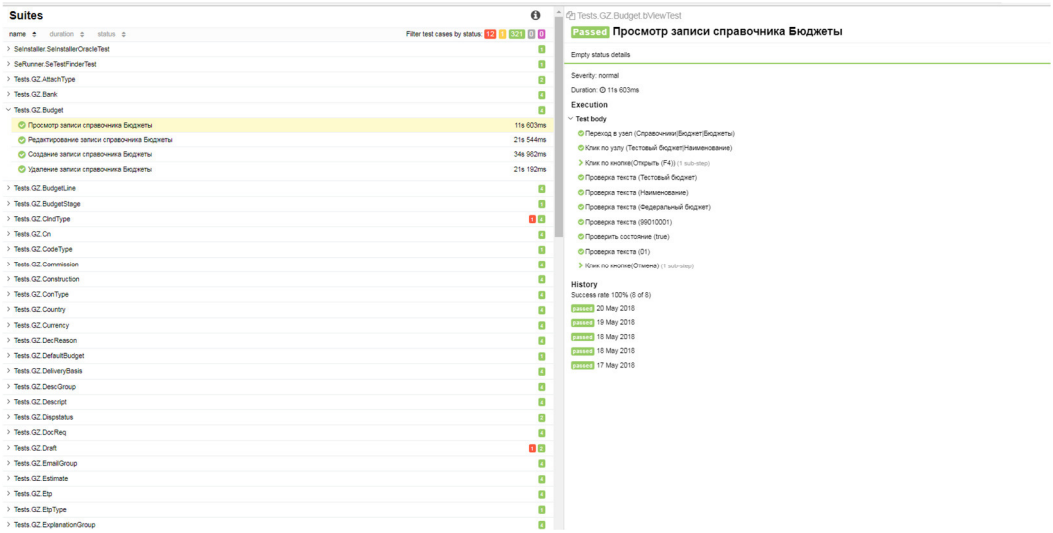

Рисунок 21 – Отчет о тестировании

3.6 Обоснование экономической эффективности

Экономическая эффективность – это результативность экономической системы, выражающаяся в отношении полезных конечных результатов ее функционирования к затраченным ресурсам [21].

Эффективность – относительный показатель результативности и может быть только положительной величиной. Экономическая эффективность – результативность экономической системы, выражающаяся в отношении полезных конечных результатов ее функционирования к затраченным ресурсам [21].

Определение экономической эффективности проекта является главным при принятии решений о рациональности инвестирования в него средств. Наряду с этим, определение эффективности инвестирования в информационные технологии часто происходит на уровне интуиции или вообще не выполняется – это вызвано нежеланием поставщиков прикладных решений использовать большие усилия на подобный анализ. С другой стороны, возможно, есть большая доля недоверия заказчиков к получаемым результатам таких исследований. Следует отметить что, обе эти проблемы происходят из одного источника, а именно – отсутствия надежных и

понятных методик оценки экономической эффективности информационно-технологических проектов.

Для определения экономической эффективности необходимо осуществить калькуляцию себестоимости. Она производится, согласно всем типовым методическим рекомендациям, утвержденным Миннауки от 15.06.1994 РФ №ОР-22-2-46 по планированию учета и калькуляции себестоимости научно-технической продукции.

Единовременные затраты на разработку – это совокупность затрат на теоретический анализ, постановку задачи, проектирование, разработку методов, алгоритмов и программ, отладку и апробацию, эксплуатацию [46].

В таблице 1 представлена фактическая трудоёмкость, распределённая по стадиям программного продукта.

Таблица 1 – Фактическая трудоёмкость

Стадия	Содержание работ	Трудоемкость, дни
Техническое задание	Изучение литературы, анализ существующих разработок и методов создания программного продукта, постановка вопросов и задач	30
Проектирование	Проектирование разработки в целом. Изучение входных и выходных данных. Обоснование выбора платформ проектируемой системы.	30
Разработка	Разработка. Написание и отладка программ.	60
Внедрение	Написание, оформление и защита дипломного проекта.	30
Итого		150

Общая трудоёмкость разработки ПО рассчитывается по формуле [37]:

$$T_{об} = \sum_{i=1}^{n} T_i = 150, \quad (1)$$

$T_{об}$ – общая трудоемкость разработки, дни;

T_i – трудоемкость (по стадиям разработки), дни;

n – количество стадий.

Разработка программы включает следующие затраты:

- материальные затраты;
- основная и дополнительная заработная плата;
- отчисления на социальные нужды;
- стоимость машинного времени на разработку программы;
- затраты на инструментальные средства;
- накладные расходы.

От себестоимости машинного часа работы ЭВМ, а так же времени работы на ЭВМ непосредственно зависит стоимость машинного времени (Зомв), а так же затраты на амортизацию оборудования, в том числе ЭВМ и на электроэнергию. Время использования оборудования рассчитывается по формуле [37]:

$$T_м = 0,35*T_{экс}+0,6*T_{тех\ пр}+0,8*T_{реб\ пр}+0,6*T_{вн} \quad (2)$$

Для данной разработки время работы на ЭВМ ($T_м$) составило 120 дней.

Расход электроэнергии вычисляются по формуле:

$$З_{эл}=C_{эл}*M_{эвм}*T_м*T_{сут} \quad (3)$$

$M_{эвм}$ − мощность ЭВМ;

$С_{эл}$– стоимость 1 кВт/ч электроэнергии, руб;

$Т_{сут}$ – время работы ЭВМ часов в сутки.

$З_{эл}$ = 3,74 руб * 0,24 кВт*120 дней*8 ч = 862 руб.

Затраты на амортизацию ЭВМ и оборудования – это затраты на приобретение оборудования и его эксплуатацию.

$$A_м = \frac{O_ф \cdot H_{ам}}{365 \cdot 100} \cdot t_{эвм}, \quad (4)$$

$А_м$ – амортизационные отчисления, руб.;

$Н_{ам}$ – норма амортизации, %;

$О_ф$ – начальная стоимость ЭВМ и оборудования, руб.;

$t_{эвм}$ – время, в течении которого использовалось оборудования, дни.

Первоначальная стоимость ЭВМ и оборудования равна 27000, тогда амортизационные отчисления составят:

$$A_м = \frac{27000 \cdot 20}{365 \cdot 100} \cdot 46 = 681 руб.$$

В итоге стоимость машинного времени равна:

$З_{омв}$ = 862 + 681 = 1543 руб.

Стоимость инструментальных средств – это совокупность стоимости системного программного обеспечения, использованного при разработке программы [37].

В таблице 2 приведено используемое для написания программы программное обеспечение, так же его стоимость.

Таблица 2 – Программное обеспечение и его стоимость

Программное обеспечение	Стоимость, руб.
TestComplete	25000

Продолжение таблицы 2 – Программное обеспечение и его стоимость

Windows 7 HomeBasic	3000
Microsoft Office 2007	12000
Итого	40000

В итоге стоимость инструментальных средств:

$$A_м = \frac{40000 \cdot 20}{365 \cdot 100} \cdot 120 = 2630 руб.$$

После завершения всех расчётов составлена таблица затрат на разработку, таблица 3.

Таблица 3 – Затраты на разработку

Наименование статей затрат	Сумма
Затраты, потраченные на оплату машинного времени	1543
Стоимость инструментальных средств	2630
Итого	4173

Итак, получаем, что затраты на создание программного средства составляют 4173 руб.

Посчитаем расходы на содержание персонала, исходя из условия, что оклад сотрудника составляет 20000 руб.

Z = 1 * 20000 * (1 + 34% / 100) = 26800 руб.

Накладные и прочие расходы до и после внедрения программы будем рассматривать как неизменные, т.е. внедрение программы не вызвало экономию чернил в картриджах принтеров, расходование бумаги и т.п. Таким образом, годовая экономия будет равна экономии, связанной с повышением производительности труда пользователя.

Рассчитаем экономию за счет увеличения производительности труда сотрудника. До внедрения автоматизированной системы тестирование ПО проводилось без использования программных средств.

Экономия, связанная с повышением производительности труда пользователя:

Р = 26800 * 9 = 241200 руб.

В итоге получаем следующую ожидаемую экономическую эффективность:

Э = 241200 – 4173 * 0,15 = 240575 руб.

Даже при приблизительном расчете экономическая эффективность от внедрения программного средства получилась значительной. Такой она получилась за счет увеличения производительности труда сотрудника.

Соответственно, потратив всего 4173 рублей, мы получаем экономию за год в 240575 рублей.

По результатам расчета экономической эффективности проектирования и внедрения средства автоматизации сразу можно сказать, что это выгодно. Хоть выгода и косвенная, но, как правило, заметная в средней и долгосрочной перспективе. Внедрение средств автоматизации может привести к корректированию самого бизнес-процесса, так как задачи выполняются быстрее. Сотрудники могут обрабатывать большие объемы информации за свое рабочее время, что можно использовать или для уменьшения затрат на персонал или для быстрого развития бизнеса при неизменности количества сотрудников, занятых обработкой информации.

Как показывает практика, автоматизация бизнес-процессов несет в себе большой потенциал для развития и материальную выгоду с течением времени.

В процессе расчета экономический эффективности необходимо учитывать одно свойство автоматизации. Заключается оно в следующем: чем больше средств и времени потрачено на автоматизацию тем выше экономический эффект от внедрения. Объясняется это просто: если качественно подойти к выбору программного продукта, качественно проработать все бизнес-процессы на этапе проектирования и внедрения, все

описать и отладить, то в последующем будет потрачено гораздо меньше средств на эксплуатацию программы.

3.7 Выводы по третьему разделу

В третьем разделе магистерской диссертации на основе модели, представленной в предыдущей главе, была разработана система автоматизированного тестирования.

Разработанная системы была протестирована с использованием программного средства АЦК-Финансы, для которого она создавалась.

Далее были произведены экономические расчеты на основании затрат, полученных в ходе разработки. С их помощью произведена оценка экономической эффективности данной разработки.

Благодаря разработанной системе повысилась эффективность работы компании за счет:

- сокращения времени процесса тестирования;
- сокращения количества специалистов по тестированию;
- уменьшения затрат денежных средств на заработную плату.

ЗАКЛЮЧЕНИЕ

В ходе исследования были выявлены множественные преимущества робота перед человеком в вопросах тестирования программного обеспечения. Выбраны методы технического анализа, которые будут положены в основу написания автотеста. Произведен анализ наиболее популярных методов.

В ходе разработки были решены следующие задачи:

– изучение теоретических данных по тестированию и автоматизированному тестированию;

– изучение основных фреймворков для написания автотестов и выбор оптимального;

– моделирование системы автоматизированного тестирования;

– программная реализация системы;

– оценка экономической эффективности данной разработки.

Результатом выполнения магистерской диссертации является разработка автоматизированной системы тестирования программного обеспечения. Благодаря данной системе повысилась эффективность работы компании, а именно:

– сократилось время процесса тестирования;

– сократилось количество специалистов по тестированию;

– уменьшились затраты денежных средств на заработную плату.

Автоматизированная система тестирования может быть использована с большой пользой, благодаря следующим ее качества:

– выполнение тестовых операций с наименьшим участием человека;

– высокая скорость обработки информации;

– возможность решения новых задач.

В настоящий момент система полностью реализована. Процесс написания тестовых сценариев максимально упрощен. Им занимается уже целая команда тестировщиков. Автоматические тесты помогают найти

большой процент программных багов и сократить трудозатраты на ручной прогон сценариев.

Продолжая тему максимальной автоматизации, в будущем планируется работа по внедрению системы автоматического создания тестовых багов по отчету о прохождении тестов. В настоящий момент для того, чтобы известить разработчиков о возникшей ошибке тестировщикам приходится вручную просматривать тестовые отчёты и создавать задачи по их исправлению. В будущем этот процесс планируется автоматизировать. Проблемы возникают лишь с определением того, на кого назначать эту работу. Обычно тестировщики сами знают, кто из разработчиков занимался той или иной функциональностью и сами решают на кого назначать работу. При автоматизированном подходе придётся немного изменить и сам процесс разработки, для того, чтобы эту информацию можно было бы извлечь непосредственно из программного кода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Chai Assertion Library [Электронный ресурс] — Режим доступа: http://chaijs.com/

2 Fowler Martin [Электронный ресурс] — PageObject, 2013 — Режим доступа: http://martinfowler.com/bliki/PageObject.html

3 GRUNT. The JavaScript Task Runner [Электронный ресурс] — Электрон. Текстовые дан. — Режим доступа:http://gruntjs.com/

4 Google. AngularJS [Электронный ресурс] — Электрон. Текстовые дан. – – Режим доступа: https://angularjs.org

5 Graham Dorothy [Электронный ресурс] — ROI of test automation: benefit and cost, 2010 — Режим доступа: http://www.dorothygraham.co.uk/downloads/generalPdfs/ProfTesterROI.pd

6 Jasmine. Behavior-Driven JavaScript [Электронный ресурс] — Режим доступа: http://jasmine.github.io/

7 JetBrains.TeamCity [Электронный ресурс] - Powerful Continuous Integration out of the box. — Режим доступа: https://www.jetbrains.com/teamcity

8 Mocha: simple, flexible, fun. [Электронный ресурс] — Режим доступа: https://mochajs.org/

9 Modern Test Case Management Software for QA and Development Teams. [Электронный ресурс] — Режим доступа: http://www.gurock.com/testrail/

10 Protractor - end-to-end testing for AngularJS. [Электронный ресурс] — Режим доступа: http://www.protractortest.org.

11 ROI в автоматизации тестирования[Электронный ресурс] — M., 2013-2018. — Режим доступа: http://bugscatcher.net/archives/2920

12 Screenshot Reporter for Protractor. [Электронный ресурс] — Режим доступа:https://www.npmjs.com/package/protractor-screenshot-reporter.

13 SonarSource. [Электронный ресурс] — SonarQube. — Режим доступа: http://www.sonarqube.org.

14 Standalone test spies, stubs and mocks for JavaScript. [Электронный ресурс] — Режим доступа:http://sinonjs.org/.

15 UWEBDRIVER I/O Selenium 2 bindings for NodeJS. [Электронный ресурс] — Режим доступа:http://webdriver.io/

16 Use JSDoc. [Электронный ресурс] — Режим доступа:http://usejsdoc.org/.

17 ltd Cucumber. — Cucumber. Simple, human collaboration. [Электронный ресурс] — Режим доступа:https://cucumber.io.

18 Автоматизация процесса тестирования при помощи методологии и инструментальных средств IBM Rational [Электронный ресурс] / В. Ематин [и др.]. Электрон. текстовые дан. –М.: SOFTWARE-TESTING.RU, 2008-2018. – Режим доступа: http://software-testing.ru/library/vendors/156-ibm-rational

19 Алексей Баранцев. Онлайн-курс «Программирование на Python для тестировщиков». [Электронный ресурс] — М., 2010-2018 — Режим доступа: https://trainings/schedule?task=3&cid=233

20 Брауде, Э. Технология разработки программного обеспечения / Э. Брауде. — Санкт-Петербург: Питер, 2004. — 655 с.

21 Вечканов, Г. Экономическая теория / Г. Вечканов, Г. Вечканова. – Москва: Эксмо, 2007. – 448 с.

22 Вигерс, К. Разработка требований к программному обеспечению / пер. с англ.. – Москва: Русская Редакция, 2004. – 576 с.

23 Винниченко, И.В. Автоматизация процессов тестирования / И.В. Винниченко — Санкт-Петербург: Питер, 2005. — 203 с.

24 ГОСТ 7.32-2001 СИБСД. Отчет о научно исследовательской работе. – Москва: Стандартинформ, 2006. – 15 с.

25 ГОСТ 2.105-95 ЕСКД. Общие требования к текстовым документам. – Москва: Стандартинформ, 2011. – 18 с.

26 ГОСТ 7.9 – 77. Реферат и аннотация. – Москва: Изд-во стандартов, 1981. – 6 с.

27 Давыдов, М. Сравнительный анализ инструментов автоматизированного функционального тестирования Web-приложений: Mercury QuickTest Pro и IBM Rational Functional Tester 6.1 [Электронный ресурс] / М. Давыдов. Электрон, текстовые дан. – М.: SOFTWARE-TESTING.RU, 2008-2018. – Режим доступа: http://software-testing.ru/library/testing/functional-testing/149--web-mercury-quicktest-pro-ibm-rational-functional-tester-61

28 Дастин, Э. Автоматизированное тестирование программного обеспечения. Внедрение, управление и эксплуатация / Э. Дастин, Д. Рэшка, Д. Пол. — Москва: ЛОРИ, 2003. — 592 с.

29 Калбертсон, Р. Быстрое тестирование / Р. Калбертсон, К. Браун, Г. Кобб. — Москва: Вильямс, 2002. — 384 с.

30 Канер, С. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений / С. Канер. — Калининград: ДиаСофт, 2001. — 544 с.

31 Лилия Сапурина. Система автоматизированного тестирования web-приложений на Python. [Электронный ресурс] — М., 2015-2018 — Режим доступа: https://github.com/liliasapurina/python_training.

32 Липаев, В. Надежность программных средств. / В.В. Липаев. – Москва: СИНТЕГ, 1998. – 358 с.

33 Липаев В.В. Тестирование программ / В.В. Липаев. – Москва: Радио и связь, 1986. – 437 с.

34 Метрики в тестировании. [Электронный ресурс] — М., 2012-2018 — Режим доступа: http://blog.shumoos.com/archives/271

35 Михнюк, Т. Ф. Охрана труда: учебник для студ. высш. учеб. завед. / Т. Ф. Михнюк – Москва: ИВЦ Минфина, 2009 — 215с.

36 Можаев, П. Средства автоматизированного тестирования / П. Можаев – – Москва: Открытые системы, 2009. — 642с.

37 Мясниченко, О. Система автоматического тестирования / О. Мясниченко — Москва: Открытые системы, 2003— 599с.

38 Новичков, А. Автоматизированное тестирование: оценка возврата инвестиций и сопутствующие риски / А. Новичков, В. Панкратов — Москва: КомпьютерПресс, 2005. – 411с.

39 О внесении изменений в отдельные законодательные акты Российской Федерации: федеральный закон от 29 дек. 2014 г. № 460-ФЗ // Собрание законодательства. – 2015. – Ст. 13.

40 О рынке ценных бумаг: Федеральный закон от 22 апр. 1996 г. № 39-ФЗ // Собрание законодательства. – 1996. – № 16 – Ст. 1918.

41 О сроках и порядке составления и представления отчетности профессиональных участников рынка ценных бумаг в Центральный банк Российской Федерации: Указание Банка России от 15 янв. 2015 г. № 3533-У // Вестник Банка России. – 2015. – Ст. 25.

42 Орлик С. Основы Программной Инженерии [Электронный ресурс]. Электрон. текстовые дан. – Режим доступа: http://swebok.sorlik.ru/

43 Орлов, С. Технологии разработки программного обеспечения / С.А. Орлов. – Санкт-Петербург: ПИТЕР, 2002. – 464 с.

44 Официальный сайт БФТ. [Электронный ресурс] – М.: Открытые системы, 2000-2018. – Режим доступа: https://bftcom.com

45 Савкин В. Принципы управления качеством программ [Электронный ресурс]. Электрон. текстовые дан. – М.: Открытые системы, 2008-2018. – Режим доступа: http://www.osp.ru/os/2008/06/5344965

46 Святослав Куликов. Тестирование программного обеспечения. Базовый курс. [Электронный ресурс] – М., Epam Systems, 2015-2018. – Режим доступа: http://svyatoslav.biz/database_book

47 Скрипкин, К. Г. Экономическая эффективность информационных систем / К. Г. Скрипкин. – Москва: Книга по Требованию, 2002. – 250 с.

48 Соммервилл, И. Инженерия программного обеспечения / пер. с англ. А.А. Минько, А.А. Момотюк, Г.И. Сингаевская. – Москва: ВИЛЬЯМС, 2002. – 624 с.

49 Тестирование и качество ПО. [Электронный ресурс] — М., 2014-2018 — Режим доступа: http://software-testing.ru/

50 Тестирование ПО с использованием инструментов HP Mercury [Электронный ресурс] / Д. Карбасов, К. Пасевич. Электрон, текстовые дан. – М.: SOFTWARE-TESTING.RU, 2008-2018. – Режим доступа: http://software-testing.ru/library/vendors/162-hp-mercury#6

51 Тестирование программного обеспечения [Электронный ресурс]. Электрон. текстовые дан. – М.: Википедия, 2010-2018. – Режим доступа: http://ru.wikipedia.org/wiki/Тестирование_программного_обеспечения

Магистерская диссертация выполнена мной совершенно самостоятельно. Все использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

«___» _____ _____ г.

_____ _____

(подпись) (Ф.И.О.)