МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное
учреждениевысшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

09.04.03 Прикладная информатика
(код и наименование направления подготовки, специальности)

Управление корпоративными информационными процессами
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему «Исследование и разработка методики тестирования платформенных бизнес-приложений»

Обучающийся Д.В. Балашов
(Инициалы Фамилия) (личная подпись)

Научный руководитель канд. тех. наук, доцент, О.В. Аникина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2022

ОГЛАВЛЕНИЕ

Введение ... 4
Глава 1 Анализ видов тестирования бизнес-приложений ... 6
1.1 Определение тестирования ... 6
1.2 Виды тестирования ... 7
1.2.1 Функциональное тестирование ... 7
1.2.2 Нефункциональное тестирование ... 16
Глава 2 Анализ методов и моделей тестирования бизнес- приложений ... 19
2.1 Методологии тестирования ... 19
2.1.1 Каскадная модель ... 19
2.1.2 Гибкая модель ... 21
2.1.3 Модель итеративного тестирования ... 26
2.1.4 V-образная модель ... 28
2.1.5 RAD модель ... 30
2.2 Методы тестирования ... 32
2.2.1 Тестирование черного ящика ... 33
2.2.2 Тестирование белого ящика ... 36
2.2.3 Тестирование серого ящика ... 39
Глава 3 Разработка методики тестирования платформенных бизнес-приложений ... 42
3.1 Постановка задачи на разработку методики тестирования ... 42
3.2 Обзорный анализ существующих методик тестирования платформенных бизнес-приложений ... 43
3.2.1 Методика тестирования ERP-систем ... 43
3.2.2 Методика тестирования бизнес-приложений на платформе SAP ... 44
3.2.3 Методика интеграционного тестирования бизнес-приложений 1С ... 45
3.2.4 Анализ представленных методик тестирования платформенных бизнес- приложений ... 46

3.3 Методика тестирования бизнес-приложений 1С8 46
Глава 4 Апробация методики тестирования платформенных бизнес-приложений 56
4.1 Апробация методики на конфигурации бизнес-приложения 1С8 56
4.2 Апробация методики тестирования на серверной части бизнес-приложения 62
Заключение 69
Список используемой литературы 70
Приложение А Сценарии тестирования модуля справочника бизнес-приложения 1С8 74

Введение

Одним из многообещающих направлений в области управления деятельностью передовых компаний и фирм считается функциональная интеграция в их ИТ-инфраструктуру сверхтехнологичных бизнес-приложений.

Под бизнес-приложением понимается программное обеспечение, которое применяется для выполнения всевозможных бизнес-задач, а также для увеличения производительности компаний и фирм.

Чтобы обеспечить высокое качество подобных бизнес-приложений нужно провести их тестирование в процессе проектирования.

Абсолютно ясно, что компании нуждаются в наиболее актуальных и свежих методах и инструментах тестирования, которые могут обеспечить проверку качества и функциональности платформенных бизнес-приложений.

Актуальность магистерской работы обосновывается необходимостью разработки методики тестирования, которая сделает процесс тестирования более эффективным.

Объектом исследования является методика тестирования платформенных бизнес-приложений.

Предметом исследования является методика тестирования платформенных бизнес-приложений.

Целью работы является разработка и внедрение методики тестирования платформенных бизнес-приложений, которая повысит эффективность данного процесса.

Для достижения цели необходимо решить следующие задачи:

– Проанализировать и оценить существующие виды тестирования программного обеспечения.

– Проанализировать и оценить методы и модели тестирования программного обеспечения.

– Разработать методику тестирования платформенных бизнес-приложений.

– Выполнить апробацию предложенной методики для повышения эффективности тестирования платформенных бизнес-приложений.

На защиту выносятся:

– Методика тестирования платформенных бизнес-приложений.

– Результаты апробации методики тестирования платформенных бизнес-приложений.

Практическая значимость магистерской работы заключается в том, что в данной работе были описаны все современные методы тестирования и разработана методика тестирования, которая позволит сократить время на тестирование и повысит эффективность результатов тестирования.

Методы исследования: методы тестирования программного обеспечения, системный анализ.

По результатам проведённого исследования опубликована 1 статья: Балашов Д.В. Методы и модели тестирования бизнес-приложений // Студенческий: электрон. научн. журн. 2022. № 16(186).

Диссертация состоит из введения, четырех глав, заключения, списка литературы и приложения.

Первая глава посвящена определению и анализу видов тестирования. Рассмотрены характеристики функциональных и нефункциональных видов тестирования. Во второй главе проводится анализ методов тестирования бизнес-приложений, в котором анализируются достоинства и недостатки методов тестирования на основе жизненного цикла бизнес-приложений. Третья глава посвящена разработке методики тестирования платформенных бизнес-приложений. В четвертой главе представлены результаты апробации разработанной методики тестирования платформенных бизнес-приложений.

В заключении подводятся итоги работы.

Работа изложена на 73 страницах и включает 27 рисунков и 2 таблицы.

Глава 1 Анализ видов тестирования бизнес-приложений

1.1 Определение тестирования

Тестирование — это процесс оценки системы или ее компонентов с целью выяснить, удовлетворяет ли она заданным требованиям или нет. Проще говоря, тестирование — это выполнение системы для выявления любых пробелов, ошибок или отсутствующих требований, противоречащих фактическим требованиям.

Согласно стандарту ANSI/IEEE 1059, тестирование можно определить как процесс анализа элемента программного обеспечения для выявления различий между существующими и требуемыми условиями (то есть дефектами/ошибками/ошибками) и для оценки функций элемента программного обеспечения.

Тестирование программного обеспечения важно, потому что, если в программном обеспечении есть какие-либо ошибки или ошибки, их можно выявить на ранней стадии и устранить до поставки программного продукта. Надлежащим образом протестированный программный продукт обеспечивает надежность, безопасность и высокую производительность, что в свою очередь приводит к экономии времени, экономической эффективности и удовлетворенности клиентов [1].

Перечислим преимущества использования тестирования программного обеспечения:

- Экономичность: это одно из важных преимуществ тестирования программного обеспечения. Своевременное тестирование любого ИТ-проекта поможет вам сэкономить деньги в долгосрочной перспективе. В случае, если ошибки обнаружены на более ранней стадии тестирования программного обеспечения, их исправление обходится дешевле.

- Безопасность: это наиболее уязвимое и чувствительное преимущество тестирования программного обеспечения. Люди ищут надежные продукты. Это помогает в устранении рисков и проблем раньше.
- Качество продукта: это обязательное требование к любому программному продукту. Тестирование гарантирует, что качественный продукт будет доставлен клиентам.
- Удовлетворенность клиентов: основная цель любого продукта - удовлетворить своих клиентов. Тестирование UI/UX обеспечивает лучший пользовательский опыт.

Основываясь на анализе работ по тестированию, был сделан вывод, что для тестирования платформенных бизнес-приложений обычно используются виды функционального и нефункционального тестирования [15, 16].

Рассмотрим данные виды тестирования.

1.2 Виды тестирования

1.2.1 Функциональное тестирование

Функциональное тестирование — это форма тестирования и процесс обеспечения качества, который помогает проверить систему или компоненты на соответствие различным функциональным спецификациям и изложенным требованиям [9, 17]. Функциональное тестирование — это тип тестирования «черного ящика», поскольку в процессе тестирования исходный код приложения не рассматривается.

Основная цель этой формы тестирования программного обеспечения — протестировать каждую функциональность приложения путем предоставления определенных входных данных и проверки выходных данных на соответствие функциональным требованиям.

Эта форма тестирования проверяет, работает ли программное обеспечение так, как ожидают пользователи. Поскольку этот тип тестирования полностью основан на спецификациях программы, он также известен как

тестирование на основе спецификаций. В этом процессе тестирования инженеры по контролю качества сосредотачиваются на проверке системы на соответствие функциональным спецификациям и проверяют функции на соответствие заданному набору пользовательских спецификаций [2].

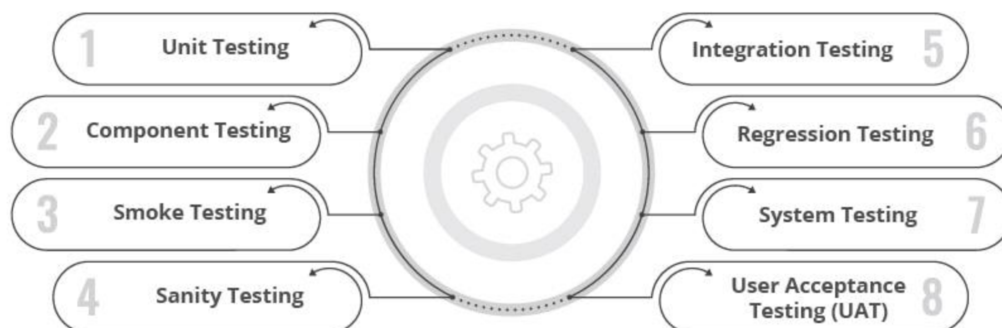

Рисунок 1 – Виды функционального тестирования.

Существуют следующие виды функционального тестирования (рисунок 1):

- модульное тестирование;
- тестирование компонентов;
- дымовое тестирование;
- санитарное тестирование;
- интеграционное тестирование;
- регрессионное тестирование;
- системное тестирование;
- пользовательское приемочное тестирование (UAT).

Рассмотрим основные из представленных видов функционального тестирования.

1.2.1.1 Модульное тестирование бизнес-приложений

Модульное тестирование — это первый этап тестирования программного обеспечения в жизненном цикле разработки программного обеспечения (STLC).

Модульное тестирование гарантирует, что разные компоненты в программе работают должным образом в соответствии с техническим заданием.

Этот тип функционального тестирования выполняется разработчиками, и они пишут сценарии для проверки того, работают ли небольшие блоки приложения перед тем как программа будет передана группе тестирования [3].

Подобное тестирование позволят сократить время, т.к. проблемы устраняются на ранней стадии.

Преимущества модульного тестирования:

- тестируя функциональность небольших модулей, можно обнаруживать ошибки до того, как они повлияют на другие модули в приложении;
- поскольку модули очень маленькие, можно запустить несколько модульных тестов за секунды;
- после рефакторинга или расширения кода можно повторно запустить все наборы тестов, чтобы убедиться, что новый или обновленный код не нарушает существующие функции;
- экономия времени разработки;
- разбивая приложение на мельчайшие тестируемые компоненты, модульное тестирование помогает увеличить покрытие кода.

Модульное тестирование может проводиться вручную, но если автоматизировать данный процесс, то можно расширить охват тестирования и ускорить циклы доставки [6], [7].

1.2.1.2 Интеграционное тестирование бизнес-приложений

Это метод тестирования программного обеспечения, при котором отдельные модули приложения объединяются и тестируются как группа для определения функциональности после объединения различных модулей.

Таким образом, интеграционное тестирование позволяет определить, действительно ли программное обеспечение, которое пишут разные разработчики, работает для достижения конечной цели [18].

Существует несколько распространённых подходов к интеграционному тестированию:

1) Подход Большого Взрыва.

Подход большого взрыва интегрирует все модули за один раз, т. е. не предусматривает интеграцию модулей один за другим (рисунок 2). Он проверяет, работает ли система должным образом или не была интегрирована. Если в полностью интегрированном модуле обнаруживается какая-либо проблема, становится трудно определить, какой модуль вызвал проблему [19].

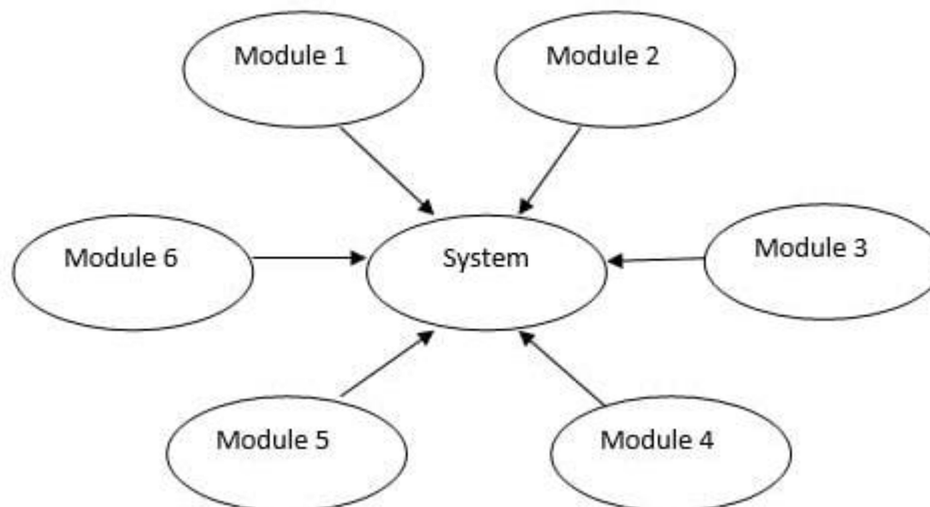

Рисунок 2 – Интеграционное тестирование по методу Большого Взрыва

Это хороший подход для небольших систем, но для крупных понадобится много времени на поиск дефекта, что может привести к потерям [20].

2) Подход снизу вверх.

Интеграционное тестирование начинается с самого нижнего модуля и постепенно продвигается к верхним модулям приложения. Эта интеграция продолжается до тех пор, пока не будут интегрированы все модули и все приложение не будет протестировано как единое целое [26], [27].

Если какие-то модули еще не были разработаны, то используются программы, которые называются драйверами.

Драйвер – это фиктивные программы, которые используются для вызова функций самого нижнего модуля в случае, когда вызывающая функция не существует. Восходящий метод требует, чтобы драйвер модуля передал входные данные тестового примера интерфейсу тестируемого модуля.

Преимущество этого подхода состоит в том, что, если в самой нижней части программы существует серьезная ошибка, ее легче обнаружить и принять меры.

Недостаток в том, что основная программа фактически не существует, пока не интегрирован и не протестирован последний модуль. В результате проблемы более высокого уровня будут обнаружены только в самом конце интеграции.

3) Нисходящий подход.

Интеграция сверху вниз — это поэтапный подход к интеграционному тестированию, при котором поток тестирования начинается с модулей верхнего уровня (модулей, находящихся выше в иерархии) к модулям более низкого уровня. Если модули более низкого уровня не были разработаны, то в этих случаях используются заглушки.

Заглушки — это фиктивные модули, которые имитируют работу модуля, принимая параметры, полученные модулем, и выдавая приемлемый результат. Как правило, заглушки имеют жестко закодированные ввод и вывод, что помогает тестировать другие интегрированные с ними модули.

Преимущество данного подхода заключается в том, что не нужно ждать, пока все модули будут разработаны. Кроме того, можно уделить

первоочередное внимание тестированию критически важных интегрированных модулей.

Одним из недостатков данного подхода является то, что требуется создание множества заглушек для имитации модулей более низкого уровня. Кроме того, модули более низкого уровня могут быть недостаточно протестированы.

Преимущества интеграционного тестирования:

- помогает выявить проблемы интеграции между модулями;
- помогает убедиться, что интегрированные модули работают правильно, прежде чем переходить к системному тестированию всего приложения;
- ошибки, обнаруженные на этом уровне, легче устранять по сравнению с обнаруженными на более поздних этапах тестирования — системном и приемочном тестировании;
- интеграционное тестирование можно начинать после того, как будут доступны тестируемые модули. Для проведения тестирования не требуется завершение другого модуля, поскольку для этого можно использовать заглушки и драйверы;
- это улучшает тестовое покрытие и обеспечивает дополнительный уровень надежности.

1.2.1.3 Системное тестирование бизнес-приложений

Это метод функционального тестирования, который выполняется после интеграционного тестирования. Этот этап тестирования системы используется для проверки полностью интегрированного программного приложения.

В основном это тестирование типа «черный ящик». Это тестирование оценивает работу системы с точки зрения пользователя с помощью документа спецификации. Это не требует каких-либо внутренних знаний систем, таких как дизайн или структура кода [8].

Чтобы протестировать систему в целом, требования и ожидания должны быть четкими.

Кроме того, наиболее часто используемые сторонние инструменты, версии ОС, разновидности и архитектура ОС могут повлиять на функциональность, производительность, безопасность, возможность восстановления или установки системы [30].

Поэтому при тестировании системы может быть полезно четкое представление о том, как приложение будет использоваться и с какими проблемами оно может столкнуться в режиме реального времени.

Основные преимущества данного метода тестирования:

- включает сквозные сценарии для тестирования системы;
- тестирование выполняется в той же среде, что и производственная среда, что помогает понять точку зрения пользователя и предотвращает проблемы, которые могут возникнуть при запуске системы;
- если это тестирование проводится систематически и надлежащим образом, это поможет смягчить проблемы постпродакшена;
- тестирование проверяет как архитектуру приложения, так и бизнес-требования.

1.2.1.4 Пользовательское приемочное тестирование бизнес-приложений

Это заключительный этап тестирования программного обеспечения, на котором конечные пользователи или клиенты берут на себя роль инженеров-испытателей, чтобы проверить, соответствует ли разработанное приложение требованиям или нет. Пользовательское приемочное тестирование — это важный тип функционального тестирования, который выполняется перед перемещением приложения в рабочую среду [31].

Пользовательское приемочное тестирование классифицируется как альфа- и бета-тестирование.

Альфа-тестирование — это когда тестирование выполняется в среде разработчика программного обеспечения и имеет более важное значение в контексте готового коммерческого программного обеспечения.

Бета-тестирование — это когда тестирование выполняется в производственной среде или в среде клиента. Это более характерно для клиентских приложений. Пользователи здесь — это настоящие клиенты.

Это тестирование играет важную роль в проверке выполнения всех бизнес-требований перед выпуском программного обеспечения для использования на рынке. Использование оперативных данных и реальных вариантов использования делает это тестирование важной частью цикла выпуска.

Можно сделать вывод, что применение разных методов тестирования для бизнес-приложений является спорной точкой для многих специалистов.

Для решения этой проблемы стоит рассмотреть один из специализированных методов тестирования бизнес-приложений.

1.2.1.5 Регрессионное тестирование бизнес-приложений

Этот дополнительный метод функционального тестирования программного обеспечения помогает убедиться, что новые изменения, внесенные в код, не влияют на уже существующие функции или функции приложения [29].

Регрессионное тестирование инициируется, когда программист исправляет какую-либо ошибку или добавляет в систему новый код для новой функциональности. Если это тестирование не будет проведено, продукт может столкнуться с критическими проблемами в реальной среде, что действительно может привести к проблемам у клиента.

Рассмотрим методы регрессионного тестирования.

1) Повторно протестировать все

Все тестовые примеры в наборе тестов выполняются повторно, чтобы гарантировать отсутствие ошибок, возникших из-за изменения кода. Это

дорогостоящий метод, так как он требует больше времени и ресурсов по сравнению с другими методами.

2) Выбор регрессионного теста

В этом методе тестовые наборы выбираются из набора тестов для повторного выполнения. Выбор тестовых случаев осуществляется на основе изменения кода в модуле.

Тестовые наборы делятся на две категории: повторно используемые тестовые наборы и устаревшие тестовые наборы. Повторно используемые тестовые случаи могут использоваться в будущих циклах регрессии, тогда как устаревшие не используются в предстоящих циклах регрессии.

3) Приоритизация тестовых случаев

Тестовые случаи с высоким приоритетом выполняются первыми, а не со средним и низким приоритетом. Приоритет тестового примера зависит от его критичности и влияния на продукт, а также от функциональности продукта, который используется чаще.

Ниже приведены различные преимущества регрессионного тестирования:

- улучшает качество продукта;
- гарантирует, что любые исправления ошибок или улучшения не повлияют на существующие функции продукта;
- для этого тестирования можно использовать средства автоматизации;
- гарантирует, что проблемы, которые уже устранены, больше не возникнут.

Несмотря на ряд преимуществ, есть и некоторые недостатки:

- тестирование необходимо делать и для небольшого изменения кода, потому что даже небольшое изменение кода может создать проблемы в существующей функциональности;
- если в проекте для этого тестирования не используется автоматизация, выполнение тестовых случаев снова и снова будет трудоемкой и утомительной задачей.

Регрессионное тестирование, по мнению специалистов, является наиболее эффективным методом для тестирования бизнес-приложений.

1.2.2 Нефункциональное тестирование

Нефункциональное тестирование — это тестирование программного приложения или системы на предмет их нефункциональных требований. Нефункциональные требования определяют работу тестируемой системы, например, как система работает, как она обслуживается и насколько она устойчива к условиям отказа. Это контрастирует с функциональными требованиями, описывающими поведение системы. Требования нефункционального тестирования часто гораздо сложнее определить, чем требования функционального тестирования, где ожидаемую функциональность и поведение легче задокументировать.

Ниже приведены наиболее распространенные типы нефункционального тестирования:

- Тестирование производительности определяет, насколько быстро приложение работает в различных ситуациях.
- Нагрузочное тестирование — это тест, который проверяет способность приложения хорошо функционировать в пиковых условиях. Подобный вид тестирования чаще всего используется для тестирования веб- приложений и клиент-серверных бизнес-приложений [28].
- Тестирование совместимости. Чтобы проверить, подходит ли оцениваемое приложение, веб-сайт или система для различных сред, таких как веб-браузеры, аппаратные платформы, базы данных, операционные системы, сети, мобильные устройства, различные версии и конфигурации.
- Юзабилити-тестирование — это вид тестирования, целью которого является определение того, насколько просто использовать программное обеспечение.

- Стресс-тестирование подвергает систему жестким условиям, таким как ее перегрузка, чтобы увидеть, сможет ли она выдержать нагрузку.
- Объемное тестирование относится к производительности программного приложения, когда оно подвергается воздействию большого объема данных.
- Тестирование безопасности должно гарантировать, что система защищает данные и продолжает функционировать должным образом.
- Тестирование надежности — это своего рода процедура тестирования программного обеспечения, которая проверяет, правильно ли работает приложение в определенных условиях в течение определенного периода времени.
- Тестирование на выносливость исследует, как система ведет себя под определенной нагрузкой в течение длительного периода времени. Мы можем оценить поведение системы, чтобы убедиться, что программное обеспечение способно выдерживать высокие нагрузки без снижения задержки.
- Тестирование документов — это процесс проверки того, что задокументированные артефакты, созданные до, во время и после тестирования продукта, являются подлинными.
- Тестирование локализации должно гарантировать, что дизайн приложения соответствует определенной культуре, региону или местному населению.
- Тестирование интернационализации должно убедиться, что приложения разработаны таким образом, чтобы они соответствовали любой культуре, региону или местному населению.
- Сравнительное тестирование — это упреждающая форма тестирования, которая используется для определения целей производительности и отслеживания прогресса с течением времени.

- Тестирование переносимости используется для проверки того, как приложение переносится из одного программного обеспечения в другое.

Функциональное и нефункциональное тестирование — это два столпа тестирования. Оба одинаково важны, когда речь идет об удовлетворении требований конечных пользователей. В то время как функциональное тестирование рассматривает все поведенческие аспекты приложения, нефункциональное тестирование обеспечивает правильную производительность в различных условиях использования.

Выводы к главе 1

Для тестирования бизнес-приложений могут использоваться функциональное и нефункциональное тестирование. Функциональное тестирование оценивает «что», тогда как нефункциональное тестирование указывает, «как» должно тестироваться приложение.

Функциональное тестирование охватывает технические функции, а нефункциональное тестирование включает в себя производительность, надежность, безопасность, масштабируемость и удобство использования.

Анализ подтвердил отсутствие общепринятых рекомендаций по применению конкретных видов тестирования для тестирования платформенных бизнес-приложений.

Глава 2 Анализ методов и моделей тестирования бизнес-приложений

2.1 Методологии тестирования

Методология тестирования программного обеспечения определяется как различные подходы, стратегии и типы тестирования для тестирования приложения, чтобы убедиться, что приложение выглядит и работает так, как ожидается, и соответствует ожиданиям пользователей/клиентов.

В широком смысле методологии тестирования включают в себя все различные типы функционального и нефункционального тестирования для проверки приложения.

Тестирование программного обеспечения является неотъемлемой частью любой методологии разработки [22]. Многие компании в просторечии используют термины «методология разработки» и «методология тестирования». Следовательно, методологии тестирования могут также относиться к моделям Waterfall, Agile и другим моделям обеспечения качества, в отличие от приведенного выше определения методологий тестирования.

Рассмотрим известные методологии тестирования, основанные на моделях жизненного цикла бизнес-приложений.

2.1.1 Каскадная модель

В каскадной модели разработка программного обеспечения проходит через различные этапы, такие как анализ требований, проектирование и т. д. — последовательно [13].

Первый этап каскадной модели — это этап требований, на котором все требования проекта полностью определяются перед началом тестирования. На этом этапе группа тестирования проводит мозговой штурм в отношении объема тестирования, стратегии тестирования и разрабатывает подробный

план тестирования. Только после того, как проектирование программного обеспечения будет завершено, команда перейдет к выполнению тестовых случаев, чтобы убедиться, что разработанное программное обеспечение ведет себя так, как ожидалось.

Этап требований гарантирует, что все необходимые требования, такие как цели тестирования, организационное планирование, проекты документов, а также стратегия тестирования определены и зафиксированы. Строгая документация и планирование делают эту модель подходящей для небольших приложений.

Затем дизайн проекта выбирается и утверждается лицом, принимающим решение. Затем команда разработчиков реализует подробный план проектирования, и после того, как он будет готов, он проверяется командой контроля качества и заинтересованными сторонами. После того, как проект проверен и запущен, команда разработчиков начинает поддерживать программный продукт до тех пор, пока он не будет тщательно протестирован для окончательного запуска.

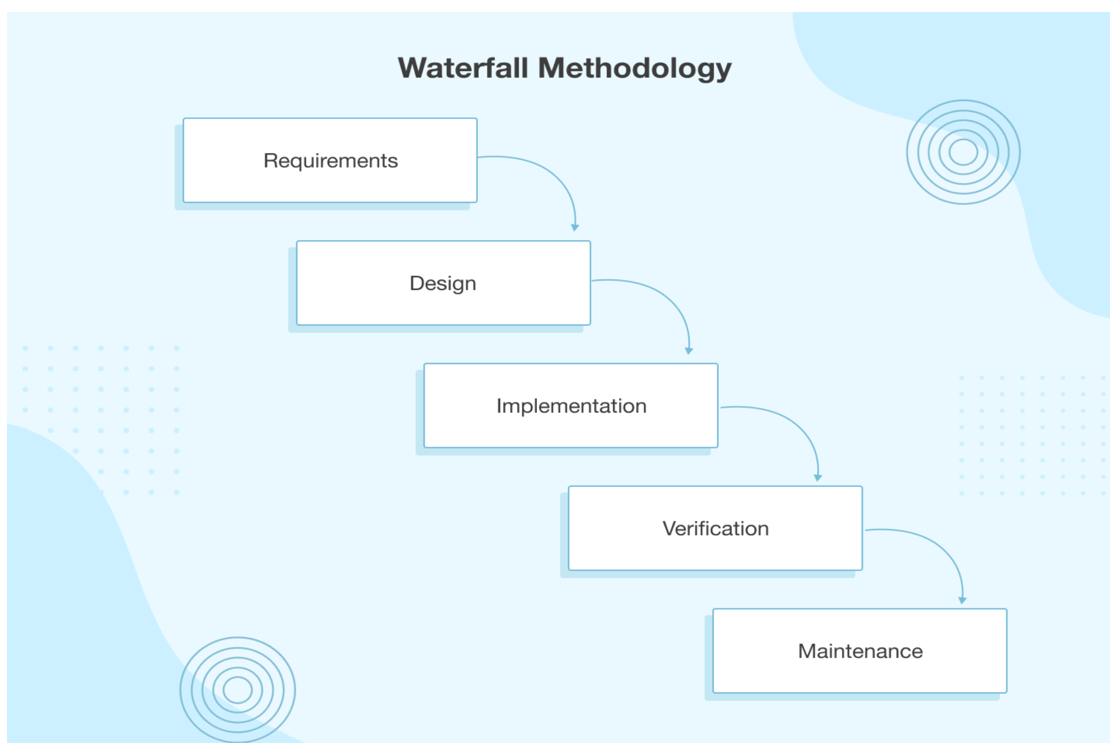

Рисунок 3 - Каскадная модель

Преимущества:

- Это пошаговая стратегия тестирования.
- Методология может быть использована для простого планирования и управления требованиями проекта.
- Это недорого и обеспечивает более быструю реализацию проектов.

Недостатки:

- Каждый шаг в методологии предопределен и не может быть пропущен.
- Он не может поглотить незапланированные итерации.
- Тестирование отодвигается на потом в списке.
- Даже небольшое изменение плана проекта может быть дорогостоящим.

Сценарии использования:

- Системы управления взаимоотношениями с клиентами.
- Системы управления персоналом.
- Системы управления цепочками поставок.
- Системы управления запасами.
- Системы торговых точек.

2.1.2 Гибкая модель

Методология гибкого тестирования основана на идее итеративной разработки, при которой прогресс в разработке осуществляется в виде быстрых инкрементальных циклов, также известных как спринты. При наличии сложных приложений и быстрых требований рынка гибкая методология открывает взаимодействие с заинтересованными сторонами, чтобы лучше понять их требования. Коммуникационный цикл позволяет команде сосредоточиться на реагировании на изменения вместо того, чтобы полагаться на обширное планирование, которое в конечном итоге может измениться.

Происхождение гибкой методологии должно было отойти от жесткой модели разработки и тестирования, не оставляющей места для итераций. Это одна из причин, по которой команда тестирования предпочитает использовать этот подход для динамических приложений, чтобы обеспечить постоянную обратную связь от заинтересованных сторон. С меньшим приоритетом для документации и большим для внедрения изменений добавочные тесты и итерации выполняются мгновенно.

Вместо того, чтобы тестировать всю систему ближе к концу, каждый предлагаемый выпуск итерации тщательно тестируется. Кроме того, каждая итерация имеет свой собственный цикл требований, цикл проектирования, кодирования и тестирования, что делает ее циклическим процессом. Модель разработки через тестирование в основном используется для добавления новых функций в существующую систему, что делает ее подходящей для небольших проектов со сжатыми сроками.

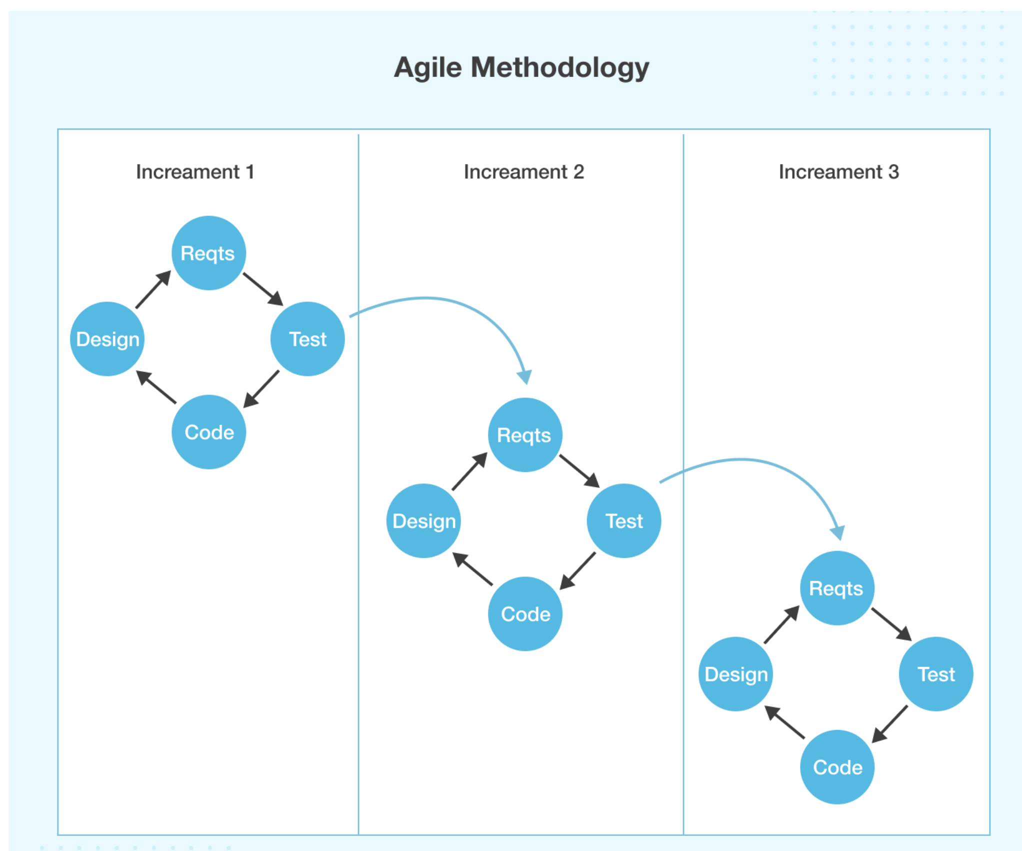

Рисунок 4 - Гибкая модель

Преимущества:

- Сложные прикладные процессы можно легко контролировать, изменять и тестировать.
- Инкрементальные тесты минимизируют затраты и риски, связанные с многочисленными изменениями.
- Постоянная связь между разработчиками и клиентами определяет прогресс программного приложения.
- Основное внимание в методологии уделяется тестированию, а не планированию.

Недостатки:

- Увеличение времени доставки туда и обратно с клиентами может привести к увеличению времени доставки.
- Меньший приоритет планирования может привести к неэффективности документации.
- Постоянные изменения усложняют обслуживание.

Сценарии использования:

- Определение области тестирования приложения.
- Новые функции в приложении.
- Тестирование нагрузки и производительности.

Гибкая методология тестирования является на сегодняшний день одной из самых распространенных методологий.

С помощью гибкой методологии Scrum рассмотрим процесс тестирования.

Scrum-тестирование — это тип тестирования программного обеспечения, которое выполняется для проверки способности программного обеспечения или приложения выполнять сложные процессы. Это тестирование также проверяет различные другие параметры программного обеспечения, такие как качество, удобство использования и производительность. Выполнение сложного процесса требует сложного

программного обеспечения. Следовательно, для создания сложного программного обеспечения требуется скрам-тестирование.

Тестировщики выполняют следующие действия на различных этапах Scrum:

1) При планировании спринта тестировщик должен выбрать из бэклога продукта пользовательскую историю, которую следует протестировать.

Тестировщик должен решить, сколько часов потребуется для завершения тестирования всех пользовательских историй.

Тестировщик должен знать цели спринта и внести свой вклад в процесс расстановки приоритетов.

2) Тестировщик поддерживает разработчиков в модульном тестировании на стадии спринта.

Выполнение теста выполняется в лаборатории, где тестировщик и разработчик работают сообща. Дефекты регистрируются и отслеживаются в специальном журнале, который постоянно контролируется и просматривается. Как только дефекты устранены, они повторно тестируются.

3) Тестировщик применяет инструменты автоматизированного тестирования.

Автоматизированное тестирование — это метод тестирования программного обеспечения, который выполняется с использованием специальных программных средств автоматизированного тестирования для выполнения набора тестовых сценариев. Напротив, ручное тестирование выполняется человеком, сидящим перед компьютером и тщательно выполняющим шаги теста.

Преимущества автоматизированного тестирования:

- автоматизированное тестирование исключает фактор человеческой ошибки. Программы, используемые для запуска тестов, не устают и не становятся невнимательными. Точность близка к 99,9%. Кроме того, почти все существующие кейсы и сценарии умещаются в несколько строк кода;

- с помощью автоматизированного тестирования можно справиться с некоторыми задачами, с которыми не справится большая команда — имитировать одновременную работу тысяч пользователей;
- выполнение того же объема работы вручную займет гораздо больше времени, тогда как автоматизированное тестирование позволяет использовать уже готовые скрипты без доработки. Таким образом, команда получает дополнительное время на тестирование проблемных зон вручную;
- одни и те же тесты можно использовать много раз — столько раз, сколько необходимо, пока не окажется, что программа не содержит ошибок. Также можно запускать несколько тестов на нескольких устройствах с разной конфигурацией, не влияя на эффективность результатов;
- ручное тестирование субъективно по сравнению с автоматическим, который генерирует статистику и отчеты после каждого теста. Они сохраняются и рассылаются по почте всем (или выбранным) членам группы;
- большие проекты с высокой нагрузкой требуют большего внимания к деталям и более тщательного контроля качества.

Рассмотрим недостатки автоматизированного тестирования:

- инструменты, используемые для автоматизированного тестирования, дороги, как и надлежащее обучение тестировщика.
- нужно некоторое время, чтобы написать автотесты, которые будут охватывать тестирование продукта.

Исходя из вышеперечисленного главной особенностью гибкой методологии является применение автоматизированного тестирования и отсутствия строго разграничения ролей в команде разработчиков.

2.1.3 Модель итеративного тестирования

Методология работает, разбивая большой проект на более мелкие компоненты, где каждый компонент проходит серию циклов тестирования. Это модель, управляемая данными, и каждая итерация основана на результатах предыдущего цикла тестирования. Повторные тесты упрощают организационное управление и оптимизируют требования к программному обеспечению перед их объединением в конечный продукт [32].

Модель итеративной разработки следует циклическому шаблону для тестирования небольших компонентов большого проекта. Каждый итерационный цикл идентичен полному циклу разработки и состоит из этапов планирования, проектирования, тестирования и оценки. После завершения цикла компонент добавляется в программное обеспечение в качестве обновления.

Когда требования к программному приложению определены слабо, обратная связь от каждой итерации улучшает функциональность конечного продукта. По этой причине модель больше всего подходит для гибких приложений с упором на масштабируемость.

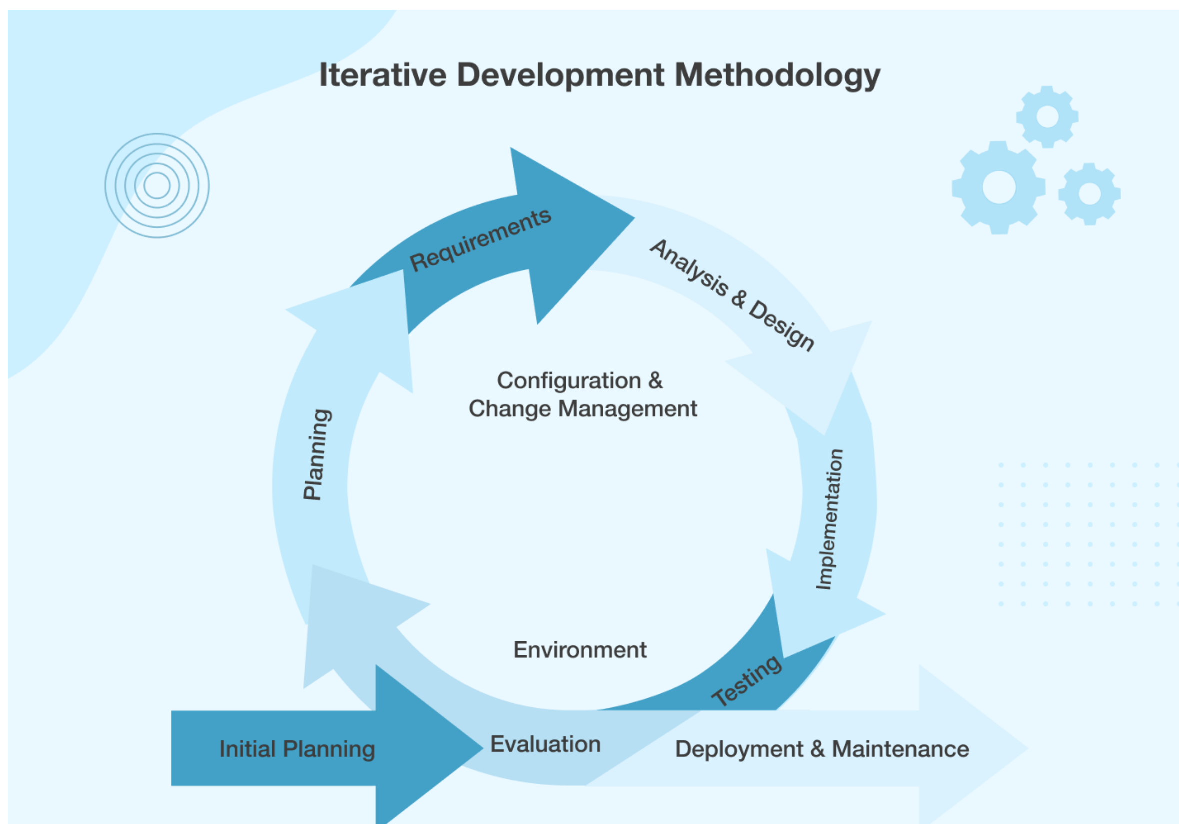

Рисунок 5 - Методика итеративного тестирования

Преимущества:

- Меньшие итерации для сложных программных приложений сокращают время и стоимость разработки.
- Обратная связь итерационных циклов доступна немедленно.
- С каждым циклом итерации устраняются ошибки и баги на начальных этапах разработки.
- Модель обеспечивает большую гибкость и фокусируется на дизайне, а не на документации.

Недостатки:

- Накладные расходы на связь могут значительно увеличиться после каждого отзыва. итерация
- Итерационный цикл является жестким и не может перекрываться.
- Непредсказуемая обратная связь может задержать выпуск конечного продукта.

- Низкий приоритет планирования может привести к непредвиденным обстоятельствам.
- Вовлечение пользователей необходимо для выявления рисков.

Сценарии использования:

- Игровые приложения.
- Потоковые приложения.
- SaaS-приложения.
- Тестирование прототипа.

2.1.4 V-образная модель

V-методология считается расширением водопадной модели, используемой для небольших проектов с определенными требованиями к программному обеспечению. Он следует V-образной модели, которая подразделяется на кодирование, проверку и проверку. Поскольку кодирование является основой модели, каждый этап разработки идет рука об руку с тестированием, что приводит к раннему обнаружению ошибок на каждом этапе.

V-модель отличается от водопадной модели параллельными процедурами тестирования, проводимыми на каждом этапе разработки. Процесс проверки гарантирует, что продукт разработан правильно, а процесс проверки гарантирует, что продукт соответствует требованиям.

Модель начинается со статических процессов проверки, которые включают анализ бизнес-требований, проектирование системы, проектирование архитектуры и проектирование модулей. Затем на этапе кодирования гарантируется, что конкретный язык и инструменты выбираются на основе стандартов и рекомендаций по кодированию. Наконец, последний этап проверки гарантирует, что каждый модуль и этап разработки проходят модульное тестирование, интеграционное тестирование, системное тестирование и тестирование приложений.

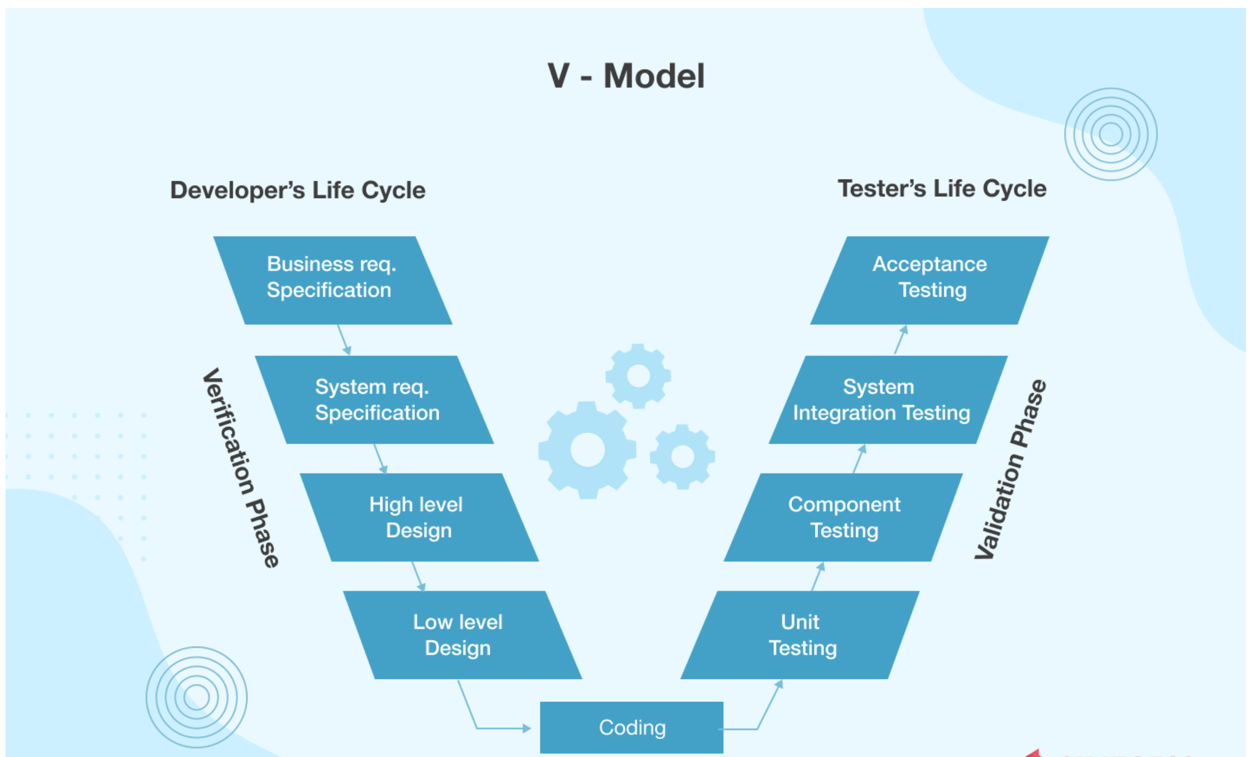

Рисунок 6 - V-образная модель

Преимущества:

- Тестирование/валидация на каждом этапе позволяет выявлять ошибки на ранних этапах цикла разработки.
- Это экономичная и эффективная по времени модель тестирования.
- Жесткость делает его идеальным для небольших проектов.
- Каждый этап валидации и верификации имеет четко определенные цели.

Недостатки:

- Нет внутренней способности реагировать на ошибки в процессе тестирования.
- Определенного решения по устранению дефекта программного обеспечения не существует.
- Модель не подходит для больших проектов с более высокими шансами на изменения.
- Он не может обрабатывать одновременные события.

- Нет пути назад после того, как модуль вошел в фазу тестирования.

Сценарии использования:

- Медицинские приборы и программные приложения.
- Государственные приложения и программные проекты.
- Оборонные проекты и приложения.
- Коммерческие приложения.

2.1.5 RAD модель

Модель тестирования разработки быстрых действий (RAD) — это форма пошаговой методологии, возникшей из гибкой системы разработки программного обеспечения. Краеугольным камнем RAD является прототипирование при параллельной разработке компонентов для программного обеспечения, поэтому основное внимание уделяется тестированию, а не планированию и документации. Хотя каждая программная функция разделена и разработана как отдельные компоненты, они объединяются для формирования прототипа, сбора отзывов конечных пользователей и соответствующих дальнейших итераций.

Методология RAD состоит из пяти этапов, на которых компоненты системы разрабатываются и тестируются одновременно. Каждый из этих этапов ограничен по времени и должен быть выполнен быстро, что делает его подходящим для проектов с жесткими сроками.

На первом этапе, «Бизнес-моделирование», определяются бизнес-требования и поток информации по другим бизнес-каналам. После того, как поток определен, на этапе «Моделирование данных» рассматриваются соответствующие данные в соответствии с бизнес-моделью.

Третий, «Моделирование процессов», преобразует объекты данных для создания потока бизнес-информации. Этап определяет процесс обеспечения качества, с помощью которого объекты данных могут быть дополнительно изменены в соответствии с отзывами клиентов. Это делается с учетом того, что приложение может проходить несколько итераций с течением времени.

Четвертый этап «Создания приложений» известен как этап прототипирования, и модели кодируются с помощью автоматизированных инструментов. Наконец, каждый прототип тестируется индивидуально на этапе «Тестирование и передача», что снижает количество ошибок и риски для всего программного приложения.

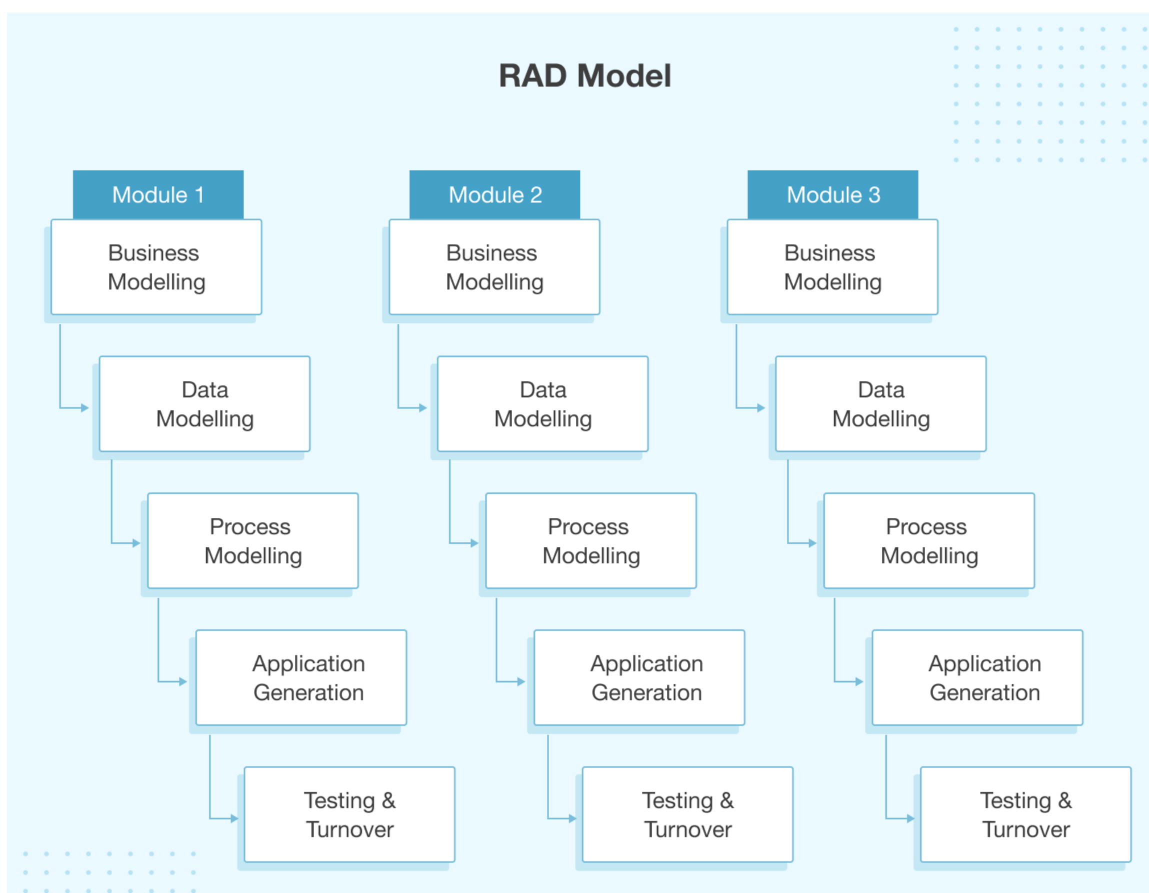

Рисунок 7 - RAD модель

Преимущества:

- Одновременное проектирование и повторное использование прототипов сокращают время разработки и тестирования.
- Подход с временными рамками на каждом поэтапном этапе снижает общие риски в программном проекте.
- Циклы обратной связи повышают удовлетворенность клиентов.
- Прогресс измерим и зависит от данных.

– Изменения можно легко приспособить.

Недостатки:

– Методология сложна для использования с устаревшими системами.
– Непрерывная обратная связь с клиентами и изменения могут задержать развертывание.
– Существует высокая зависимость от технических навыков и ресурсов.
– Автоматизированное тестирование, инструменты и генерация кода приводят к более высоким затратам.

Сценарии использования:

– Улучшение графического пользовательского интерфейса приложения.
– Прототипы приложений (Wireframe, Design и Clickable прототип)
– Модульность системы.

Существует множество методологий разработки программного обеспечения и его соответствующего тестирования. Каждая методика и методология тестирования предназначена для определенной цели и имеет свои относительные достоинства и недостатки.

Выбор конкретной методологии зависит от многих факторов, таких как характер проекта, требования клиента, график проекта и т. д.

С точки зрения тестирования, некоторые методологии настаивают на тестировании входных данных на ранних этапах жизненного цикла разработки, в то время как другие ждут, пока не будет готова рабочая модель системы.

2.2 Методы тестирования

Сегодня существует множество подходов к тестированию программного обеспечения, но наиболее популярными в этой области являются методы тестирования «черный ящик», «белый ящик» и «серый ящик». Несмотря на

совершенно разные подходы, эти методы эффективно помогают разработчикам содержать код в чистоте и контролировать его функциональность.

2.2.1 Тестирование черного ящика

Что характерно для метода тестирования черного ящика, так это то, что тестировщики, выполняющие его, не знают внутренней структуры и исходного кода тестируемого программного обеспечения [21].

Тестирование «черного ящика» можно назвать функциональным тестированием или тестированием на основе спецификаций.

На рисунке 8 представлена модель тестирования по методу черного ящика.

Рисунок 8- Модель тестирования по методу черного ящика

Такие тесты выполняются с точки зрения конечных пользователей независимой группой тестирования в ходе STLC. Тестер предоставляет допустимые или недопустимые входные данные и сверяет выходные данные с

ожидаемыми результатами. Каждый неожиданный результат и отклонение документируются и сообщаются команде разработчиков, что помогает им находить и устранять функциональные ошибки и несоответствия на раннем этапе.

Этот метод применим практически на всех уровнях тестирования программного обеспечения: модульное, интеграционное, системное и приемочное. В модульном тестировании метод черного ящика используется для проверки интерфейса на соответствие спецификациям, заданным клиентом.

Целью интеграционного тестирования является поиск и устранение ошибок во взаимодействии интегрированных компонентов интерфейса. Метод «черного ящика» также можно эффективно применять при тестировании системы для анализа соответствия системы требованиям, а также при приемочном тестировании, где он может помочь подтвердить приемлемость программного продукта путем его тестирования в различных непредвиденных ситуациях и обстоятельствах.

Некоторые из наиболее распространенных методов разработки тестов «черного ящика» включают в себя:

- Тестирование таблиц решений удобно при отладке программного обеспечения на основе встроенных операторов if-then-else и switch-case в таблицах решений. Это эффективный способ найти ошибки в том, какие действия каким условиям соответствуют.
- Угадывание ошибок означает, что тестер разрабатывает тестовые примеры на основе своей интуиции и опыта предыдущего тестирования. Они используют его, чтобы определить, что может привести к сбою программного обеспечения или появлению ошибок.
- Тестирование всех пар — это метод, используемый для проверки всех возможных дискретных комбинаций каждой пары входных параметров. Это действительно помогает найти распространенные

ошибки, которые обычно кроются во взаимодействиях между парами параметров.

- Метод разделения эквивалентности включает в себя разделение входных данных на разные более мелкие разделы, классы эквивалентных данных, из которых могут быть получены тестовые примеры. Этот метод используется для создания тестового примера, который охватывает каждый раздел сразу, что сокращает время, необходимое для тестирования.

Тестирование методом «черного ящика» действительно может помочь выявить любую двусмысленность, расплывчатость и противоречия в функциональных спецификациях. Это позволяет тестировщикам оценивать и повышать качество реализации функциональности, не вмешиваясь напрямую в большие сегменты кода программного обеспечения.

Тестирование методом «черного ящика» совершенно беспристрастно, потому что тестирование проводится независимой командой, которая отделяет точку зрения конечных пользователей от точки зрения разработчиков. Среди трех методов тестирование методом «черного ящика» характеризуется самой быстрой разработкой тестового примера, поскольку оно не требует знаний в области программирования и может быть легко выполнено тестировщиками без технического образования.

Тем не менее, этот метод можно эффективно применять только для тестирования небольших частей программного обеспечения. Тщательное тестирование большого сложного программного обеспечения с использованием этого метода оказалось бы довольно неэффективным, а также потребовало бы очень много времени. Кроме того, этот метод требует четких и исчерпывающих спецификаций, чтобы быть эффективным. В противном случае будет крайне сложно разработать тестовые примеры, а сценарии обеспечат очень ограниченный охват.

Таким образом, при тестировании бизнес-приложений с помощью метода «черного ящика», следует учесть следующее:

- необходимо обеспечить корректность входных данных;
- необходимо обеспечить участие экспертов, которые знают особенности реализуемых функций.

2.2.2 Тестирование белого ящика

В отличие от тестирования методом черного ящика, которое фокусируется на функциональности, цель метода тестирования белого ящика состоит в том, чтобы выполнить анализ внутренней структуры программного обеспечения и лежащей в его основе логики. Поэтому тестирование белого ящика иногда называют структурным тестированием или тестированием, управляемым логикой. Этот метод требует очень много времени и требует от тестировщиков сильных навыков кодирования, полного знания программного обеспечения, которое они тестируют, и доступа ко всем исходным кодам и документам по архитектуре, иначе тестировщики не смогут разработать надлежащие тестовые примеры.

Следовательно, тестирование методом белого ящика обычно выполняется профессиональными разработчиками, которые применяют свой опыт, чтобы получить внутреннее представление о структуре, выяснить, что именно происходит в исходном коде, и исправить то, что работает не так, как ожидалось. В дополнение к глубоким знаниям метод также требует специализированных инструментов для анализа и отладки исходного кода.

Тестировщики тщательно изучают код и другие внутренние аспекты данного программного обеспечения, определяют все допустимые и недопустимые входные данные, а затем сверяют выходные данные с ожидаемыми результатами. Они проверяют операторы и условия, пути кода и потоки данных, чтобы убедиться в отсутствии скрытых ошибок или элементов, подверженных дефектам (рисунок 9).

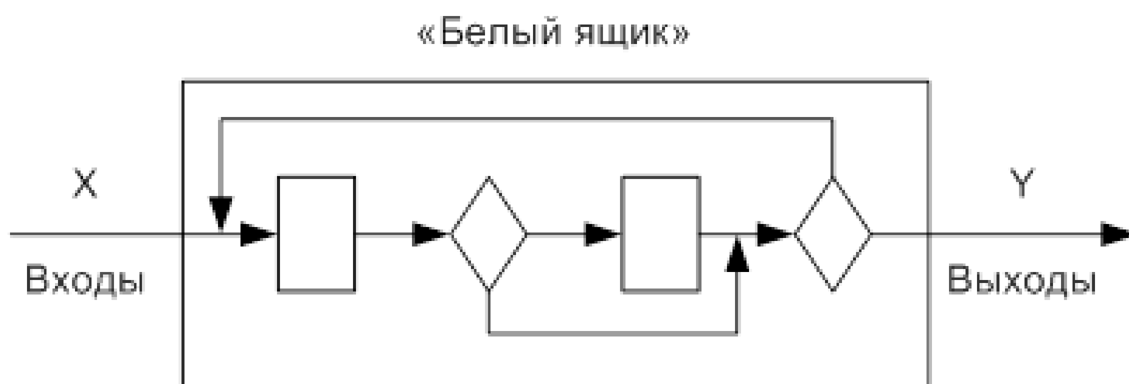

Рисунок 9 - Модель тестирования по методу белого ящика

Тестирование белого ящика в бизнес-приложениях может применяться на уровне модульного тестирования. Это делается до того, как произойдет какая-либо интеграция с ранее протестированным кодом, что снижает риск появления ошибок на более поздних этапах разработки.

В интеграционном тестировании метод помогает анализировать взаимодействия между различными интерфейсами и подсистемами. Во время регрессионного тестирования метод белого ящика можно очень эффективно применять за счет использования тестовых случаев белого ящика, переработанных на уровне модульного и интеграционного тестирования.

Некоторые из наиболее распространенных методов проектирования тестов «белого ящика» включают в себя:

- Тестирование потока управления — это стратегия структурного тестирования, в которой используются потоки управления программного обеспечения для проверки логики кода путем выполнения входных значений и проверки их соответствия требуемым результатам.
- Тестирование потока данных выявляет неправильное использование значений данных и аномалии потока данных, вызванные ошибками кодирования. Этот метод направлен на обнаружение сомнительных областей кода, чтобы можно было провести дополнительное тестирование для исправления или устранения этих ошибок.

- Не всегда существует один непрерывный поток кода. Иногда код разветвляется для выполнения определенных функций, охватывающих разные условия истинности/ложности. Техника тестирования ветвей фокусируется на проверке этих ветвей и устранении аномалий.

По сравнению с тестированием методом «черного ящика» метод «белого ящика» похож на точечный удар, который выявляет ошибки в скрытом коде, удаляя лишние строки. Такое глубокое знание исходного кода облегчает работу с побочными эффектами, что очень полезно. Это также позволяет отслеживать каждый тест на уровне исходного кода, где каждое будущее изменение может быть легко зафиксировано во вновь добавленных или модифицированных тестах.

Он выявляет все незаметные узкие места в коде, предоставляет команде разработчиков максимальное освещение и четкую и краткую обратную связь. Это облегчает команде разработчиков сокращение технического долга за счет оптимизации и поддержания качества своего кода, тем более что тестирование белого ящика также можно автоматизировать.

Однако, автоматизированное или нет, тестирование белого ящика обычно требует очень много времени и является сложным. Этот подход требует от тестировщиков первоклассных навыков программирования и полного понимания тестируемого программного обеспечения на уровне кода. Это подразумевает наем первоклассных инженеров для того, чтобы тесты были эффективными, что также удорожает метод.

Результаты тестирования также строго привязаны к способу написания кода. Если изменить код, связанный с той же функциональностью, предыдущие предположения становятся недействительными, что может привести к неудачному тесту с ложными срабатываниями. Кроме того, в то время как метод «черного ящика» эффективен при функциональном тестировании, тестирование «белого ящика» не может быть эффективным, поскольку оно фокусируется только на существующем состоянии

программного обеспечения. Это означает, что он не сможет предоставить какие-либо отзывы об отсутствующей функциональности, оставив многие пути непроверенными.

2.2.3 Тестирование серого ящика

Метод серого ящика расширяет охват методов тестирования, фокусируясь на всех слоях тестируемого программного обеспечения, независимо от его сложности. В то время как тестировщики «черного ящика» проверяют, все ли в порядке с интерфейсами и функциональностью, а тестировщики «белого ящика» вникают во внутреннюю структуру и исправляют исходный код программного обеспечения, тестирование «серого ящика» занимается и тем, и другим одновременно, ненавязчиво. способ.

Метод серого ящика предназначен для сложных систем с простым подходом черного ящика, который позволяет выполнять тесты практически всем, от разработчиков до тестировщиков и конечных пользователей. Однако для разработки тестовых случаев инженеру требуется частичное знание внутренней структуры, включая документацию по структурам данных, архитектуре, а также функциональным спецификациям программного обеспечения. Сгенерированные тестовые примеры направлены на поиск и устранение дефектов в структуре и закрытие любых пробелов, которые могут привести к неправильному использованию программного обеспечения.

Тестирование серого ящика оказывается наиболее полезным на уровне интеграционного тестирования. Он хорошо подходит для тестирования веб-приложений, поскольку у них нет исходного кода или двоичных файлов, что делает невозможным их тестирование методом белого ящика. Тестирование серого ящика также может быть применено к тестированию бизнес-домена, чтобы подтвердить, что программное обеспечение соответствует требованиям.

Некоторые из наиболее распространенных методов проектирования тестов серого ящика включают в себя:

- Матричное тестирование отслеживает и отображает требования пользователей, чтобы убедиться, что в тестовых примерах учтено все. Это позволяет легко определить недостающую функциональность. Это похоже на отчет о состоянии, подтверждающий полноту покрытия тестами.
- Регрессионное тестирование — это, по сути, анализ влияния изменений в программном обеспечении. Он включает в себя проверку правильности работы программного обеспечения после модификаций. Этот метод используется, чтобы убедиться, что нет новых ошибок и ничто не препятствует существующей функциональности.
- Метод тестирования шаблонов анализирует ранее обнаруженные дефекты в сборке, дизайне и архитектуре тестируемого программного обеспечения. Этот анализ применяется для поиска основной причины, конкретной причины данного дефекта и предотвращения его повторения.

Метод тестирования «серого ящика» сочетает в себе преимущества тестирования «черного ящика» и тестирования «белого ящика», избавляя от большинства их недостатков. Этот метод очень ненавязчив, поскольку он основан на функциональных спецификациях, интерфейсах и документации, которые дают тестировщикам возможность заглянуть в архитектуру программного обеспечения, а не получить полный доступ к исходному коду или двоичным файлам.

Это также означает, что существует четкая граница между тестировщиками и разработчиками, что делает этот метод тестирования абсолютно беспристрастным. Кроме того, тестирование серого ящика позволяет создавать интеллектуальные тесты — исключительные тестовые сценарии для анализа типов данных, коммуникационных протоколов, исключений и т. д.

Однако этот метод требует отличного управления проектом, поскольку тестирование может быть избыточным, если разработчик уже выполнил соответствующие тестовые случаи. Из-за ограниченного знания внутренней структуры программного обеспечения и отсутствия доступа к его исходному коду тестирование серого ящика обеспечивает лишь частичное тестовое покрытие с непротестированными многими путями кода, что также делает его непригодным для тестирования алгоритмов. И последнее, но не менее важное: очень сложно связать идентификацию дефектов в распределенных системах с использованием метода серого ящика.

Исходя из вышеизложенного можно сделать вывод, что единая методология тестирования бизнес-приложений отсутствует [14].

Для тестирования платформенных бизнес-приложений более целесообразно использовать Agile-методологии тестирования благодаря применению автоматизированных механизмов тестирования.

Выводы к главе 2

– Тестирование методом черного ящика использует требования для получения внешних ожиданий и устранения функциональных ошибок и несоответствий.

– Тестирование методом белого ящика исследует исходный код, чтобы убедиться в отсутствии скрытых ошибок или элементов, подверженных дефектам.

– При тестировании серого ящика используются высокоуровневые данные и функциональные спецификации для обнаружения дефектов и проверки соответствия программного обеспечения требованиям.

– Отсутствует единая методология тестирования бизнес-приложений. Для тестирования платформенных бизнес-приложений более целесообразно использовать Agile-методологии тестирования благодаря применению автоматизированных механизмов тестирования.

Глава 3 Разработка методики тестирования платформенных бизнес-приложений

3.1 Постановка задачи на разработку методики тестирования

Изучение некоторого количества источников информации подтвердило отсутствие общего подхода к оценке эффективности тестирования бизнес-приложений.

Из критериев эффективности необходимо подчеркнуть минимизацию «человеческого фактора» в процессе проведения тестов бизнес-приложений. Это крайне важно, если точность, а главное корректность выполнения является критической для бизнеса [12].

Одной из возможностей снижения человеческого фактора является автоматизация процессов тестирования приложений для бизнеса. Однако на ряду с этим, требуется обеспечить полное, функциональное тестирование приложений. Для постановки задач на создание методики проведения тестирования платформенных бизнес-приложений, требуется рассмотреть их функциональные, а также архитектурные особенности. Рассмотрение будет проводиться на примере решений платформы 1С8. Модель взаимодействия компонентов и «клиент-серверная» архитектура платформы бизнес-приложения на базе 1С8 изображена на рисунке 10 [4, 5].

Рисунок 10 – Клиент-серверная архитектура бизнес-приложения на платформе 1С8

Одной из важнейших задач приложений, созданных для бизнеса является обработка транзакций в реальном времени (OLTP), она определяет повышенные требования к производительности серверных платформ.

Учитывая вышеизложенное формулируются требования к методике тестирования:

- обеспечить проверку приложения с учетом архитектуры «клиент-серверной»;
- обеспечить проверку производительности серверов на которых будут базироваться бизнес- приложения;
- использовать механизмы автоматизации технологической платформы для всех этапов тестирования.

Если применять методику, построенную с учетом вышеизложенных требований, то есть вероятность повысить эффективность процесса тестирования без снижения качества.

3.2 Обзорный анализ существующих методик тестирования платформенных бизнес-приложений

3.2.1 Методика тестирования ERP-систем

Приступая к обзорному анализу, необходимо рассмотреть методики тестирования, в которых требуется воспроизвести несколько видов тестирования, обеспечивающих проверку функционала различных функций системы [16, 23].

Обозначенная методика представляет собой несколько этапов:

1) Функциональное тестирование. Основанная на принципах точного списка целей и определений, этот метод тестирования обеспечивает, что каждая из функций будет работать и отвечать требованиям организации.

2) Тестирование производительности - данный тип проверяет, стабильность работы и полноту функционирования ERP-решения при взаимодействии с различными подсистемами, с которыми оно должно взаимодействовать.

Проверка действий с высоким уровнем потока данных, таких как транзакции, которые должны выполняться даже при высоких пиковых нагрузках. Собственно, цель тестирования – дать гарантии, что ERP-система является устойчивой стрессовым нагрузкам.

3) Интеграционное тестирование – данный вид тестирования должен дать подтверждение, что в ERP-систему включены все необходимые процессы, которые должны выполняться. Проводится как тестирование отдельных модулей ERP-системы, так и группы.

В таком тестировании используются реальные сценарии, где реальные пользователи тестируют штатные для системы ситуации, с которыми им приходится сталкиваться работе. Целью является - удостовериться, что все компоненты и модули в ERP-системе работают.

Важно производить тестирование производительности и функциональное тестирование совместно на ранних этапах внедрения.

Средства автоматизации рекомендуется использовать для повышения эффективности тестирования.

Отсутствие в методике указаний по применению конкретных методов и моделей на каждом этапе является главным недостатком и ограничивает ее возможности.

3.2.2 Методика тестирования бизнес-приложений на платформе SAP

Для обеспечения автоматизации используется решение от компании IBM (Rational Functional Tester) [25].

Представленная методика не так требовательна к уровню подготовки тестировщика.

Благодаря этому начинающий тестировщик или специалист с малым стажем может создавать сценарии функционального тестирования, генерируемые как упрощенные сценарии.

QA-инженеру не потребуются знания в программировании для корректировки сценариев тестирования. Можно использовать визуализацию

элементов приложения для добавления контрольных точек, команд, и дополнительных элементов для тестирования.

3.2.3 Методика интеграционного тестирования бизнес-приложений 1С

Данная методика основана на принципах повышения эффективности интеграционного тестирования приложений разработанных на базе платформ 1С. Применяется следующий подход: для веб-сервиса, который может подвергаться изменениям разрабатывается набор так называемых (smoke) тестов или «дымовых» тестов. Данные виды тестов проверяют, что основные элементы сервиса не подвергались изменениям и обращение к ним не вызовет ошибок.

В интеграционном тестировании используется механизм программных заглушек. Такой сервис дает возможность переопределить существующий сервис и подставить вместо него упрощенный вариант реализации, которая работает так, как это необходимо разработчику и предоставляет доступ к собственным вариантам данных и настройкам.

Для реализации мок-сервисов стоит использовать программное обеспечение SoapUI, оно позволяет создать мок-сервис на основании WSDL-схемы и разместить его на частном веб-сервере.

Приложение 1С над которым проводится тестирование не станет обращаться к удаленному сервису, а получит ответ, заранее подготовленный и опубликованный разработчиком из SoapUI (рисунок 11).

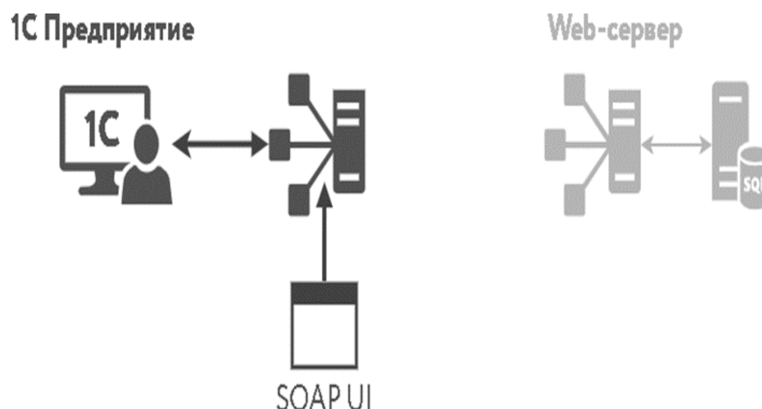

Рисунок 11 – Схема процесса тестирования

Данная методика имеет ограничения, ведь предназначена для тестирования приложений 1С, интегрированных с веб- сервисами по технологии СОА.

3.2.4 Анализ представленных методик тестирования платформенных бизнес- приложений.

Анализ представленных методик тестирования позволяет сделать определённые выводы:

1) Основным минусом рассмотренных методик является их привязка к используемым видам и методам тестирования к архитектуре технологических платформ, на базе которых реализуется бизнес- приложение.

Методики тестирования платформенных бизнес- приложений не универсифицированны.

2) Тестирование не должно быть трудоемким и финансово затратным, к уровню навыков тестировщика не должны предъявляться высокие требования, его участие в процессе должно по возможности быть ограничено. Последнего возможно достигнуть, применяя средства автоматизации.

3) Упор в указанных методиках тестирования, производится на функциональное тестирование, которое, проводится на модульных и системных уровнях.

Из этого следует, что для повышения показателей эффективности при тестировании платформенного бизнес-приложения требуется применять методику, разрабатываемую с учетом архитектурных особенностей платформы и имеющую ориентацию на применение средств автоматизации.

3.3 Методика тестирования бизнес-приложений 1С8

Опираясь на исследование известных методик тестирования бизнес-приложений, была разработана методика тестирования, для приложений, созданных на базе технологической платформы «1С: Предприятие 8».

Тестирование бизнес-приложения на базе платформы 1С8 выполняется

согласно следующим этапам:

1) Тестирование конфигурации приложения.

Подразумевает системный, а также модульный уровни тестирования.

2) Тестирование серверной части бизнес-приложения.

Представляет собой функциональное тестирование СУБД, а также нагрузочное тестирование серверов.

UML диаграмма вариантов использования данной методики проведения тестирования отображена на рисунке 12.

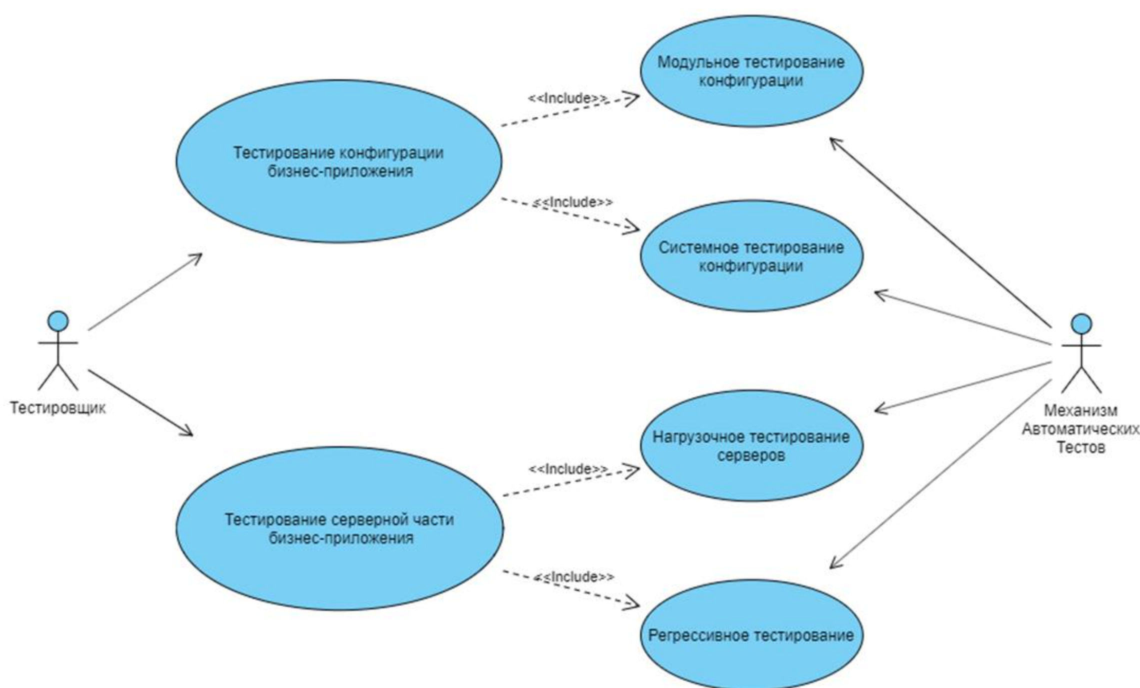

Рисунок 12 – Диаграмма вариантов использования методики тестирования бизнес-приложений 1С8

Тестирование конфигурации - это тестирование, которое сопоставляет бизнес-приложение с некоторым количеством комбинаций программно-аппаратного обеспечения, чтобы определить конфигурации, в которых система может работать без ошибок.

Тестирование конфигурации проводит проверку поведения приложения

в определенных средах.

Данный вид тестов показывает непосредственное влияние изменений в конфигурациях на производительность систем.

Число вариантов конфигураций на этапе проектирования может быть слишком большим для тестирования. Этот факт дает понять, что непосредственно на этапе планирования необходимо определить конфигурации, которые будут поддерживаться в дальнейшем.

Приоритеты устанавливаются на основе пользовательской базы и рисках, сопряженных со скрытыми ошибками в конфигурациях.

Это, достаточно, длительный процесс для тестирования, он должен проводиться после любого изменения в конфигурации, и занимает значительное время для установки и удаления требуемого ПО, которое используется в тестировании.

В источниках используется понятие базовой конфигурации (рисунок 13).

Тест конфигураций позволяет обнаружить не критичные для функционирования прикладных решений ошибки, но при этом, наличие которых оказывает пагубное влияние на производительность работы решения или приводить к появлению критичных ошибок в работе при использовании определенных режимов.

Рисунок 13 – Пример тестируемых конфигураций

Проведение подобных проверок делается, для проверки конфигурации перед поставкой для заказчика, или перед выпуском массового решения, или после объединения конфигураций [10].

Функциональное тестирование конфигурации в 1С8 включает в себя как модульное, так и системное тестирование.

Тестирование конфигурации 1С8 в этом случае выполняется методом тестирования по сценарию.

Тестирование бизнес-приложения на платформе 1С8 включает:

- проверку логической целостности конфигурации;
- выявление некорректных или пустых ссылок;
- синтаксический контроль модулей;
- логическую проверку модулей и др.

Для полноценного проведения функционального тестирования бизнес-приложения необходимо применять методы автоматизированного тестирования технологической платформы.

Тестирование серверной части включает в себя несколько уровней:

1) Тестирование сервера баз данных (СБД).

СБД – это вариант СУБД, которая используется сервером бизнес-приложения.

Для функционального тестирования СБД требуется применять модель тестирования реляционной СУБД, разработанную С. Амблером [24].

Модель выстроена с в виде единой базы данных информационных систем.

Пунктирными линиями обозначены границы угроз, которые требуется учитывать, как изнутри БД при тестировании методом белого ящика, так и на уровнях интерфейсов с базой данных, при тестировании по методу черного ящика (рисунок 14).

По мнению экспертов Agile есть несколько причин, по которым требуется создать комплексную методику тестирования СУБД:

- данные - важный корпоративный ресурс;

– в СУБД содержится критичная для бизнеса функциональность;

– некоторые методы разработки, например, такие как рефакторинг, опираются на идею, необходимости наличия возможности определять, повреждались ли данные в БД при внесении изменений.

Рисунок 14- Модель тестирования СУБД

В таблице 1 показаны объекты функционального тестирования СУБД.

Для тестирования серверной СУБД применяется методика (Test-Driven Development, TDD) -Разработка через тестирование.

TDD - это новый подход к разработкам, сочетающий в себе метод Test-First Development, TFD – (разработки в первую очередь) содержащий

тестирование и рефакторинг.

В данном контексте, рефакторингом является процесс изменения функционирующего кода бизнес-приложения.

Таблица 1 - Объекты функционального тестирования СУБД

Тестирование интерфейса (метод черного ящика)	Внутреннее тестирование базы данных (метод белого ящика)
Отображение мета-данных; Значения входных данных; Значения выходных данных запросов, представлений и хранимых процедур.	Типовые модульные тесты для хранимых процедур, триггеров и функций; Существующие тесты для элементов базы данных, таких как, таблицы и процедуры; Определения представлений; Ссылочная целостность; Значения по умолчанию для столбца; Варианты данных для одного столбца; Варианты данных, включающие несколько столбцов

Алгоритм такого метода представлен в виде диаграммы UML на рисунке 15.

Для увеличения эффективности тестирования необходимо использовать механизмы автотестов.

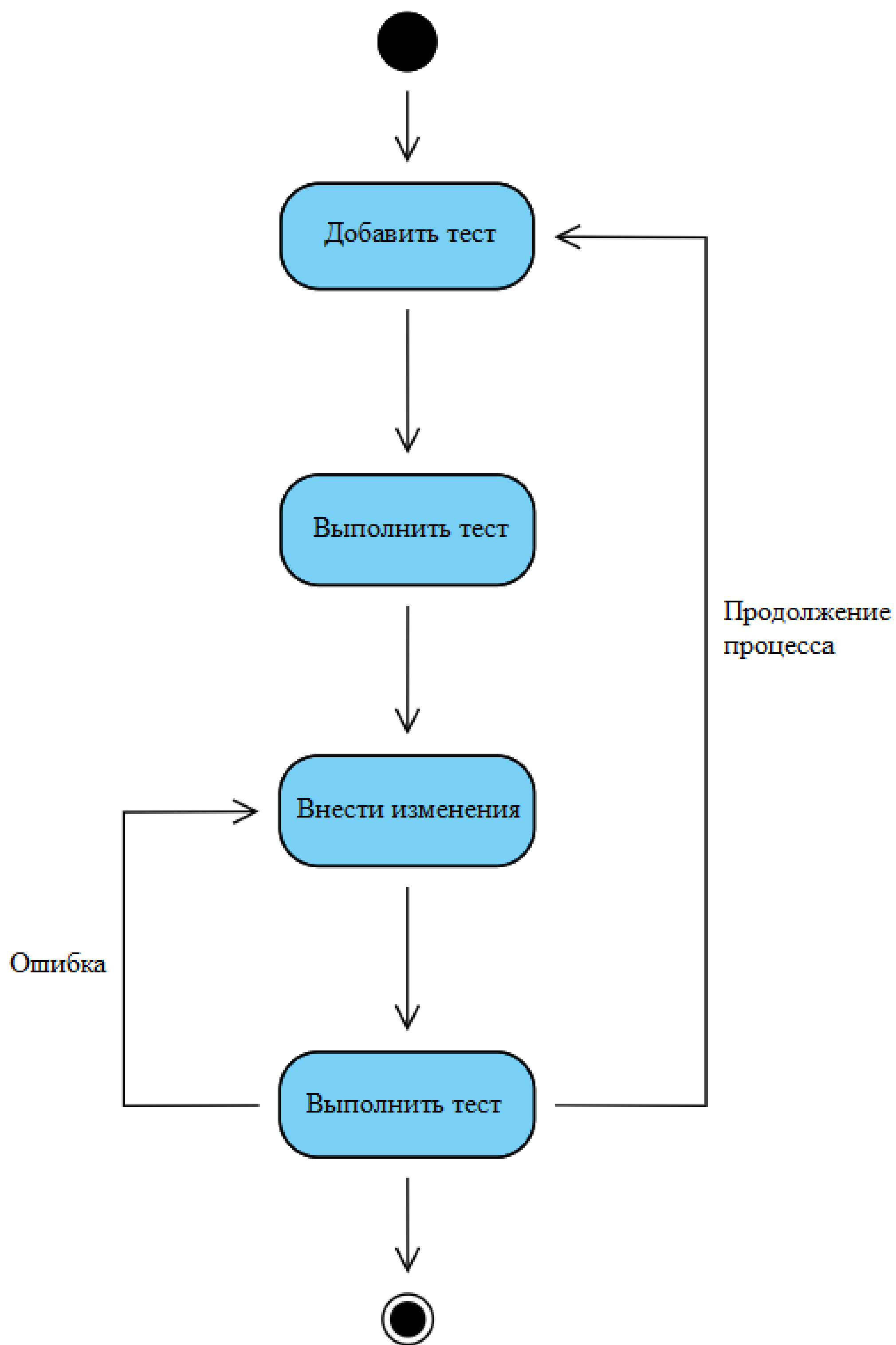

Рисунок 15 – Алгоритм метода TDD

2) Нагрузочное тестирование серверной части бизнес-приложений. Крайне важной деталью в любом внедрении или изменении

информационной системы является оценка быстродействия систем, скорости их отклика и расчет необходимых технических мощностей для ее реализации.

Существует достаточно большое количество способов оценки необходимых конфигураций программного и аппаратного обеспечения для достижения требуемого уровня производительности. Эти способы можно применять в процессе подбора, но при этом потребитель должен понимать их область применения и возможно, заложенные ограничения.

Для решения подобных задач используется, нагрузочное тестирование. Нагрузочное тестирование серверных платформ это один из обязательных видов тестирования входящий в комплекс методики тестирования платформенного бизнес-приложения.

Обусловлено это тем, что промышленные технологические платформы, часто, могут работать с несколькими СУБД.

Как пример, платформа «1С: Предприятие 8.3» поддерживает несколько СУБД: Microsoft SQL Server, PostgreSQL, IBM DB2, а так же Oracle Database.

При выборе определенной СУБД или переходе с одной СУБД на другую из перечня поддерживаемых, результаты нагрузочных тестов будут являться одними из ключевых для принятия решения.

Другим крайне значимым объектом для тестирования будет является сервер приложений «1С: Предприятия 8» или по-другому, просто «Сервер 1С».

Крайне важно учитывать, что современные бизнес-приложения на базе платформы 1С8 работают в режиме терминального доступа.

В таком случае проведение нагрузочного тестирования может помочь разработчикам оценить уровень производительности внедряемых терминальных решений.

Процесс нагрузочного тестирования обычно представляет собой следующие этапы:

- Анализ требований и сбор информации о тестируемой системе;
- Конфигурация тестового стенда для нагрузочного тестирования;

- Разработка модели нагрузки;
- Выбор инструмента для нагрузочного тестирования;
- Создание и отладка тестовых сценариев;
- Тестирование;
- Анализ результатов;
- Подготовка, отправка и публикация отчета о нагрузочном тестировании.

Перечисленные этапы не могут быть проведены параллельно или в другом порядке. Они выполняются с учетом результатов предыдущих этапов.

Нагрузочное тестирование серверов для платформенных приложений создается на базе методов авто-тестирования, созданных для данной платформы.

Это приводит к значительному росту эффективности процесса.

Для проведения нагрузочных тестов серверной части в 1С8, необходимо использовать нагрузочный авто-тест TPC-1C, именуемый иначе, как Гилева [11].

В его основу заложен метод оценки производительности, основанный на том какое количество операций в единицу времени способна выполнить система.

В данном тесте применяются два основных метода тестирования: компонентный и интегральный.

При компонентном тестировании проводится тестирование отдельных компонентов решения, производительность процессов, подсистемы хранения процессов и тестирование производительности сервера в целом, но не применяю полезную нагрузку в виде того или иного приложения.

Характеристикой интегрального подхода, является оценка производительности решения, включая программную и аппаратную часть.

Может быть использовано как бизнес-приложение, которое должно использоваться в конечном решении, так и определенные модельные приложения, эмулирующие бизнес-процессы и нагрузку. Настройки СУБД, и

настройки операционных систем, а также оборудование, все это оказывает влияние на общий совокупный результат.

Тест призван оценить количество работы выполняемой в единицу времени на одном потоке, но также подходит для оценки скорости работы однопоточных нагрузок, например, скорость графического интерфейса. Данный тест относят к разделу универсальных кроссплатформенных тестов. Он применим даже для клиент-серверного и файлового вариантов реализаций.

Работает на всех СУБД, поддерживаемых 1С, его универсальность позволяет совершать оценку производительности, без привязки к конкретной конфигурации платформы.

Ключевыми недостатками теста является то, что он не проводит диагностики, а также не оценивает возможностей оборудования для коллективной работы в условиях интенсивных блокировок.

Выводы к главе 3

Чтобы повысить эффективность тестирования платформенного бизнес-приложения должна применяться методика, в которой будут рассмотрены реализация бизнес-приложения в клиент-серверной архитектуре и высокие требования к производительности серверов.

Данная методика тестирования бизнес-приложения включает в себя комплекс - тестирование конфигурации бизнес-приложения совокупно с его серверной частью.

Предложенная методика основывается на использовании механизмов автоматического тестирования «1С: Предприятие 8».

Рекомендуется использовать регрессионное тестирование TDD для функционального тестирования СУБД.

Нагрузочное тестирование серверов в «1С: Предприятие 8» следует производить с использованием автоматизированного нагрузочного теста ТРС-1С.

Глава 4 Апробация методики тестирования платформенных бизнес-приложений

Апробация разработанного метода была проводилась на базе технологической платформы «1С: Предприятие 8.3», работающей в связке с СУБД PostgreSQL-PRO.

4.1 Апробация методики на конфигурации бизнес-приложения 1С8

В ходе эксперимента, в стандартную конфигурацию был добавлен новый справочник.

На рисунке 16 представлен алгоритм тестирования конфигурации бизнес-приложения на базе платформы 1С8.

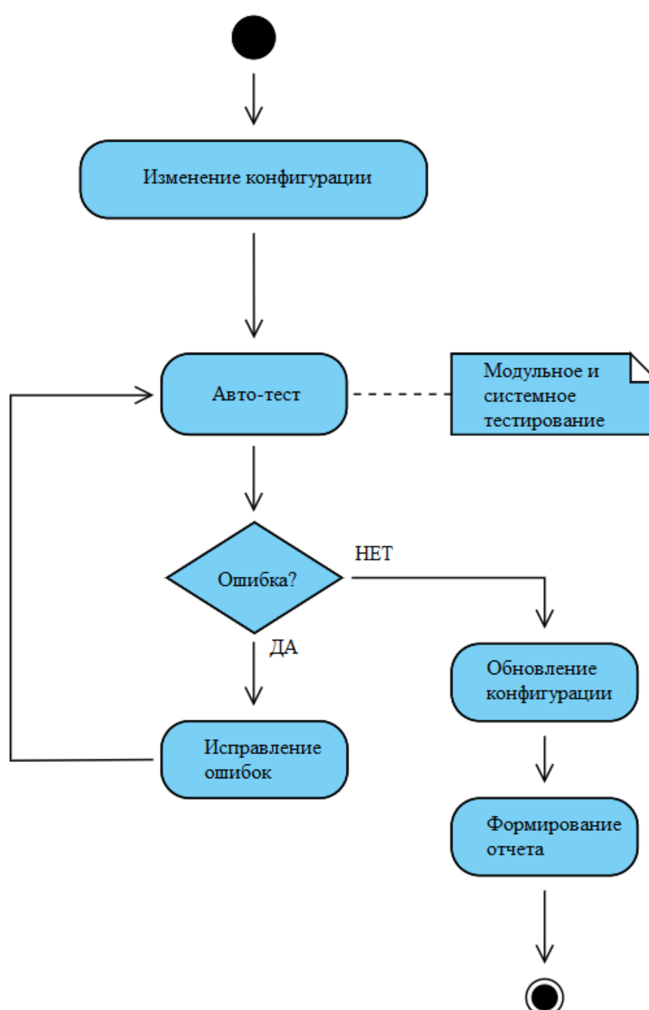

Рисунок 16- Алгоритм тестирования конфигурации

С целью автоматизации процессов тестирования конфигураций используются встроенные в платформу механизмы автоматизации, модель механизма автоматизации изображена на рисунке 17.

Рисунок 17 - Модель механизма автоматизированного тестирования платформы

Чтобы автоматизировать тестирование, обычно используются такие клиентские приложения как - менеджер и клиент тестирования. Менеджер устанавливает связь с клиентом и осуществляет предопределенный сценарий теста.

Сценарий теста – ни что иное как код на встроенном в платформу языке, в котором описывается ход этапов интерактивных действий, которые должны выполняться. Для реализации такой возможности во встроенный язык были добавлены объекты, которые описывают интерфейс на абстрактном уровне приложения, а также описывают и имитируют пользовательские действия.

Менеджер тестирования может представлять собой толстый или тонкий

клиент, а вот клиентом тестирования – может являться как толстый/тонкий клиент, так и веб версия клиента.

Автоматизированное тестирования выполняется согласно алгоритму, состоящему из определенных шагов:

Шаг 1. Создать сценарий тестирования – разработать обработку конфигурации встроенную или внешнюю, в которой должны быть описаны выполняемые этапы тестирования строго соблюдая последовательность.

Шаг 2. Запустить менеджер тестирования.

Шаг 3. Выполнить запуск клиентов тестирования.

Шаг 4. В менеджере запускается для выполнения созданная обработка. Обязательно требуется убедиться, что на клиенте выполняются запрограммированные действия.

На рисунках 18-20 представлен пример сценария теста, который выполняет проверку версии конфигурации согласно заданному списку требований.

```
#Если Сервер ИЛИ ТолстыйКлиентОбычноеПриложение ИЛИ
ВнешнееСоединение Тогда
#Область Обработчики

// Обработчик перед записью документа
// Синхронизирует конфигурацию по версии.
//
Процедура ПередЗаписью(Отказ, РежимЗаписи, РежимПроведения)
		Если ОбменДанными.Загрузка Тогда
		Возврат;
	КонецЕсли;
		Если Версия.Владелец <> Конфигурация Тогда
		Конфигурация = Версия.Владелец;
	КонецЕсли;
КонецПроцедуры

// Выполняет проверку конфигурации в соответствии с
// переданным списком требований.

Процедура ОбработкаПроведения(Отказ, РежимПроведения)

	// Определяем последний номер версии
	ЗапросПоНомеру = Новый Запрос;
	ЗапросПоНомеру.Текст = "
	|ВЫБРАТЬ
	|	ВерсииОбъектов.НомерВерсии КАК НомерВерсии,
	|	ВерсииОбъектов.Объект КАК Объект
	|ИЗ
	|	РегистрСведений.ВерсииОбъектов КАК ВерсииОбъектов
	|ГДЕ
	|	ВерсииОбъектов.Объект = &Объект
	|
	|СГРУППИРОВАТЬ ПО
	|	ВерсииОбъектов.Объект,
	|	ВерсииОбъектов.НомерВерсии
	|ИТОГИ
	|	МАКСИМУМ(НомерВерсии)
	|ПО
	|	Объект";
		ЗапросПоНомеру.УстановитьПараметр("Объект", Ссылка);

	ВыборкаНомеров = ЗапросПоНомеру.Выполнить().Выбрать();

	Если ВыборкаНомеров.Следующий() Тогда
		НомерВерсии = ВыборкаНомеров.НомерВерсии + 1;
```

Рисунок 18 – Сценарий проверки конфигурации

```
	Иначе
		НомерВерсии = 1;
	КонецЕсли;

	// Формируем версию в регистре версий
	НоваяЗапись =
РегистрыСведений.ВерсииОбъектов.СоздатьМенеджерЗаписи();
	НоваяЗапись.ДатаВерсии = ТекущаяДатаСеанса();
	НоваяЗапись.Объект = Ссылка;
	НоваяЗапись.НомерВерсии = НомерВерсии;
	НоваяЗапись.ВерсияОбъекта = Новый
ХранилищеЗначения(СформироватьТабличныйДокумент());
		НоваяЗапись.Записать();
	КонецПроцедуры
#КонецОбласти
#Область ПрочиеПроцедурыИФункции
// Возвращает представление документа в виде табличного документа.

Функция СформироватьТабличныйДокумент()
		// Делаем выборку ошибок
	НомераОшибок = Ошибки.ВыгрузитьКолонку("Номер");
		ЗапросПоОшибкам = Новый Запрос;
	ЗапросПоОшибкам.Текст = "
	|ВЫБРАТЬ
	|	НайденныеОшибки.Объект.Путь КАК Объект,
	|	НайденныеОшибки.Номер КАК Номер,
	|	НайденныеОшибки.Ошибка.Наименование КАК Ошибка,
	|	НайденныеОшибки.Состояние КАК Состояние,
	|	НайденныеОшибки.Ответственный КАК Ответственный,
	|	НайденныеОшибки.МестоОбнаружения КАК
МестоОбнаружения,
	|	НайденныеОшибки.Уточнение КАК Уточнение
	|ИЗ
	|	РегистрСведений.НайденныеОшибки КАК НайденныеОшибки
	|ГДЕ
	|	НайденныеОшибки.Номер В(&НомераОшибок)";

	ЗапросПоОшибкам.УстановитьПараметр("НомераОшибок",
НомераОшибок);

	Выборка = ЗапросПоОшибкам.Выполнить().Выбрать();
		ТабДокумент = Новый ТабличныйДокумент;
	МакетОформления =
ПолучитьМакет("ЗаданиеНаИсправлениеОшибок");
```

Рисунок 19 – Сценарий проверки конфигурации (продолжение)

```
	Заголовок = СтрШаблон(НСтр("ru='Задание на исправление ошибок №%1 от %2'"), Номер, Дата);
		МакетОформления.Параметры.Заголовок = Заголовок;
	МакетОформления.Параметры.Конфигурация = Конфигурация;
	МакетОформления.Параметры.Версия = Версия;
	МакетОформления.Параметры.Ответственный = Ответственный;

	ТабДокумент.Присоединить(МакетОформления.ПолучитьОбласть("Заголовок"));

	Пока Выборка.Следующий() Цикл
		МакетОформления.Параметры.Объект = Выборка.Объект;
		МакетОформления.Параметры.Номер = Выборка.Номер;
		МакетОформления.Параметры.Ошибка = Выборка.Ошибка;
		МакетОформления.Параметры.Состояние = Выборка.Состояние;
		МакетОформления.Параметры.Место = Выборка.МестоОбнаружения;
		МакетОформления.Параметры.Уточнение = Выборка.Уточнение;
	ТабДокумент.Присоединить(МакетОформления.ПолучитьОбласть("Ошибка"));
			КонецЦикла;
		ТабДокумент.Показать();
		Возврат ТабДокумент;
	КонецФункции

#КонецОбласти

#КонецЕсли
```

Рисунок 20 – Сценарий проверки конфигурации (окончание)

Сценарий проведения тестов выполнен в виде внешних обработок, который хранится в файле с расширением *.EPF (рисунок 21).

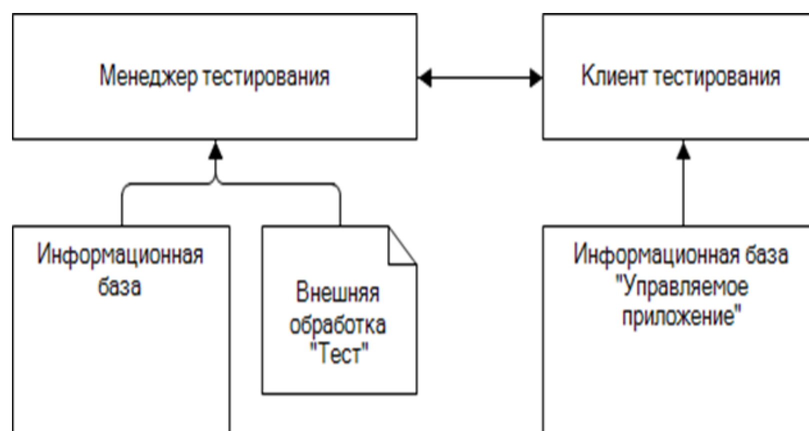

Рисунок 21 – Модель авто-теста, выполняемого с помощью внешней обработки

Результат модульного тестирования справочника бизнес-приложения (Приложение А), представлены в таблице ниже (таблица 2).

Таблица 2 – Отчет по результатам модульного тестирования справочника

№	Описание	Результат
1	Тест на соответствие проекту разработки	Соответствует заявленным требованиям
2	Модульное тестирование	Ошибки устранены
3	Системное тестирование	Нет ошибок
4	Тест качества	Соответствует заявленным требованиям

Разработчики бизнес-приложений, создают библиотеки тестовых сценариев и в дальнейшем используют их в работе.

4.2 Апробация методики тестирования на серверной части бизнес-приложения

На схеме 22 указан алгоритм, по которому проводится тестирование информационной базы.

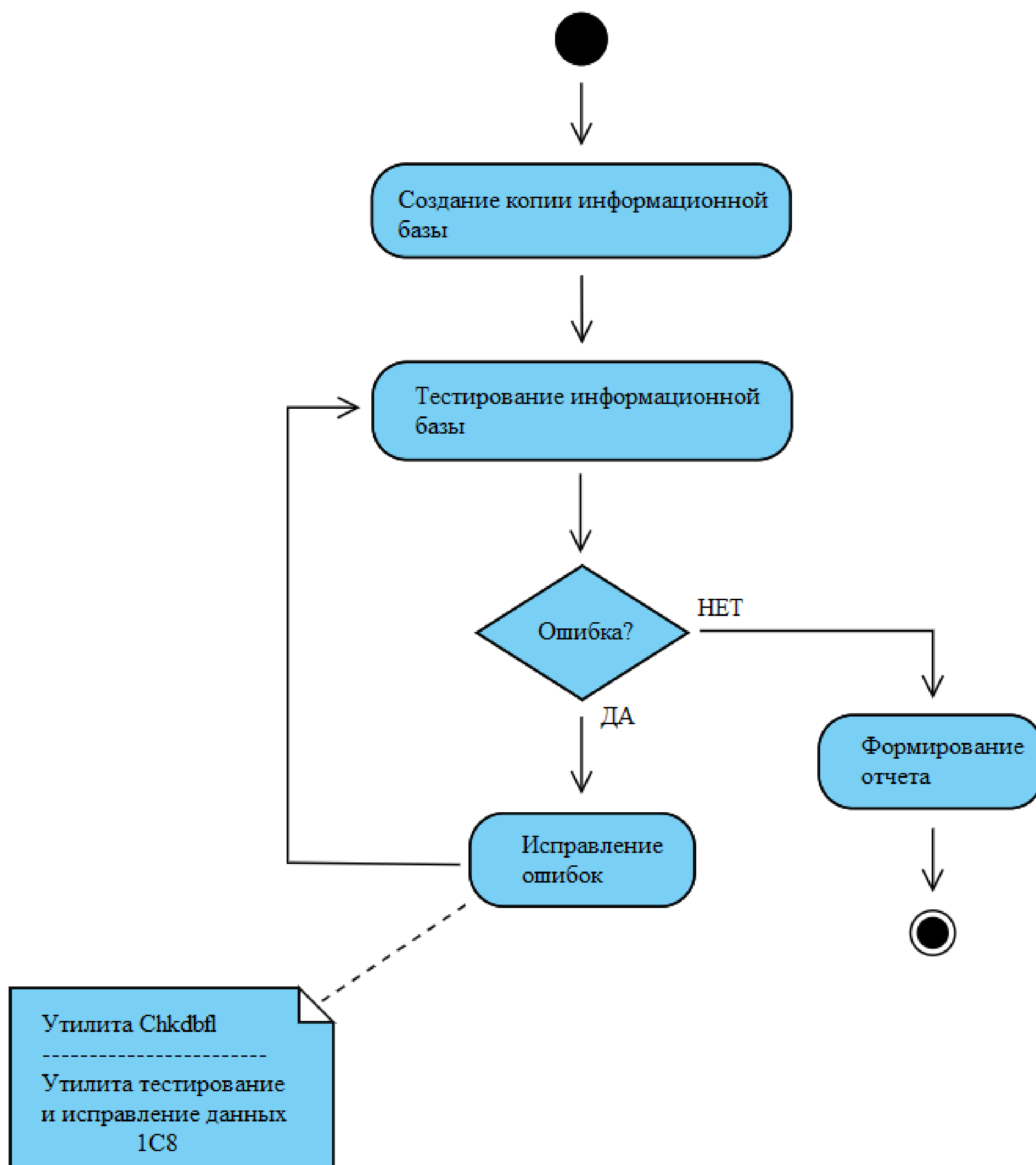

Рисунок 22 – Алгоритм тестирования информационной базы.

Для автоматизации процесса используется модуль «Тестирование и исправление» встроенная в конфигуратор платформы (рисунок 23).

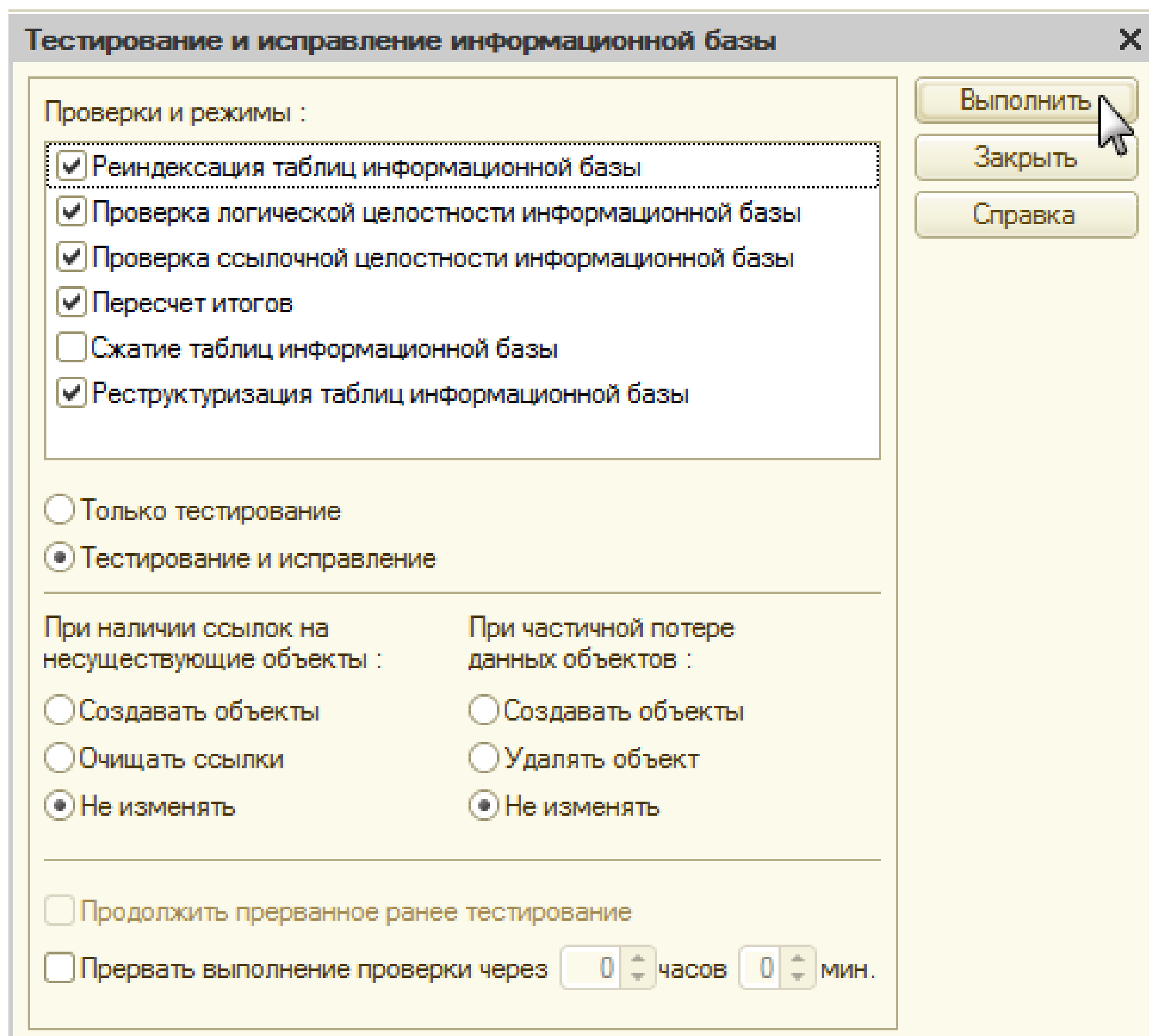

Рисунок 23 – Модуль «Тестирование и исправление»

Данный модуль реализует следующие способы проверки информационной базы:

– проверка логической целостности - цель проверка логики базы данных;

– реиндексация таблиц - за счёт перестроения индексов таблиц обеспечивается быстродействие базы данных;

– проверка ссылочной целостности – тесты, в процессе которых выявляются «пустые ссылки»;

– пересчёт итогов — с помощью данной проверки проводится пересчёт итогов таблиц базы данных;

- реструктуризация таблиц ИБ;
- сжатие таблиц базы (для файловой ИБ).

Используя тест TPC-1C было произведено нагрузочное тестирование СУБД, терминального сервера и сервера приложений 1С. Для автоматизации был использован набор инструментов, представленный на сайте разработчика.

На рисунках 24-26 представлены нагрузки серверов приложения.

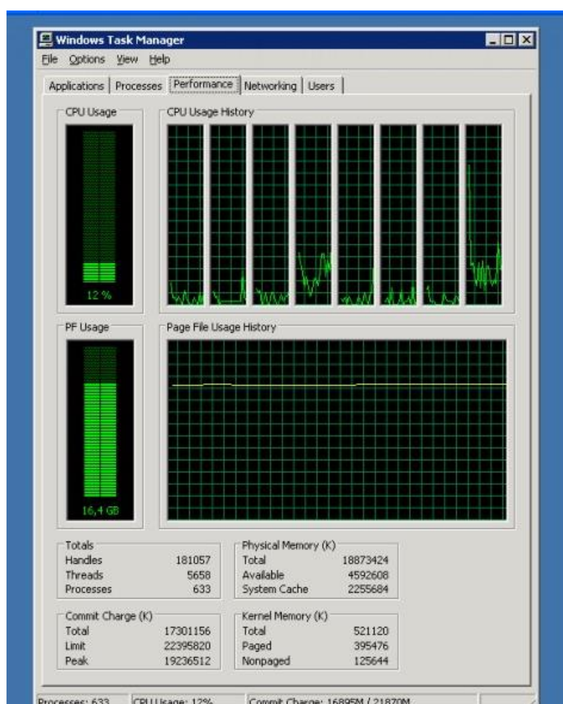

Рисунок 24 – Обычная нагрузка сервера (20 пользователей)

```
top - 11:19:10 up 11:05,  1 user,  load average: 1.44, 1.22, 1.12
Tasks: 125 total,   1 running, 124 sleeping,   0 stopped,   0 zombie
Cpu0  :  8.0%us,  0.3%sy,  0.0%ni, 91.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  : 37.2%us,  1.3%sy,  0.0%ni, 58.1%id,  1.7%wa,  0.3%hi,  1.3%si,  0.0%st
Cpu2  :  2.3%us,  0.3%sy,  0.0%ni, 97.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  2.7%us,  0.7%sy,  0.0%ni, 96.3%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Mem:   8313504k total,  1640656k used,  6672848k free,    82460k buffers
Swap:  5668856k total,        0k used,  5668856k free,   633104k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3348 usr1cv82  15   0  782m 397m  58m S 31.6  4.9  17:03.74 /opt/1C/v8.2/i386/rphost -range 1560:1591 -reghost 192.168.100.204 -regport
 3347 usr1cv82  15   0  763m 348m  58m S 17.3  4.3  44:07.53 /opt/1C/v8.2/i386/rphost -range 1560:1591 -reghost 192.168.100.204 -regport
10545 usr1cv82  15   0  382m 145m  56m S  0.0  1.8   0:11.28 /opt/1C/v8.2/i386/rphost -range 1560:1591 -reghost 192.168.100.204 -regport
10536 usr1cv82  15   0  377m 140m  55m S  0.0  1.7   0:09.59 /opt/1C/v8.2/i386/rphost -range 1560:1591 -reghost 192.168.100.204 -regport
 3334 usr1cv82  15   0  231m  24m  13m S  4.7  0.3   4:52.26 /opt/1C/v8.2/i386/rmngr -port 1541
 3331 usr1cv82  15   0  113m  12m 9128 S  0.0  0.2   0:01.99 /opt/1C/v8.2/i386/ragent -daemon
 3618 root      34  19 26620  10m 2188 S  0.0  0.1   0:00.29 /usr/bin/python -tt /usr/sbin/yum-updatesd
 3191 root      15   0 14464 4796 1068 S  0.0  0.1   0:00.00 python ./hpssd.py
 3603 haldaemo  18   0  6248 4336 1700 S  0.0  0.1   0:01.32 hald
 3289 ntp       15   0  4288 4288 3304 S  0.0  0.1   0:00.01 ntpd -u ntp:ntp -p /var/run/ntpd.pid -g
10453 root      15   0 10132 2868 2288 S  0.0  0.0   0:00.06 sshd: root@pts/0
```

Рисунок 25 – Нагрузка сервера 1С

```
top - 11:25:14 up 11:06,  1 user,  load average: 0.73, 0.99, 0.89
Tasks: 174 total,   4 running, 170 sleeping,   0 stopped,   0 zombie
Cpu0  : 29.6%us,  2.7%sy,  0.0%ni, 66.1%id,  1.3%wa,  0.0%hi,  0.3%si,  0.0%st
Cpu1  : 20.3%us,  8.7%sy,  0.0%ni, 71.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  : 20.5%us,  0.3%sy,  0.0%ni, 78.5%id,  0.7%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  0.0%us,  0.3%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu4  :  6.7%us,  0.0%sy,  0.0%ni, 93.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu5  : 14.7%us,  1.0%sy,  0.0%ni, 84.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu6  :  3.0%us,  0.0%sy,  0.0%ni, 97.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu7  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  32959992k total, 32704788k used,   255204k free,    73676k buffers
Swap:  2031608k total,      172k used,  2031436k free, 28841980k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 9014 postgres  16   0 2751m 2.6g 2.0g R   67  8.2   9:05.75 postmaster
 9250 postgres  16   0 2752m 2.6g 2.0g R   22  8.2   4:40.04 postmaster
 8026 postgres  15   0 2728m 2.6g 2.0g S    8  8.1  12:28.85 postmaster
 8189 postgres  16   0 2304m 2.1g 2.0g R    8  6.8  14:47.33 postmaster
 3546 postgres  15   0 81868 3496  280 S    2  0.0   4:21.43 postmaster
 3804 root      15   0  234m  25m 9888 S    1  0.1   6:24.58 java
10782 root      15   0 12716 1128  804 R    0  0.0   0:00.10 top
    1 root      18   0 10324  688  580 S    0  0.0   0:02.02 init
    2 root      RT  -5     0    0    0 S    0  0.0   0:00.02 migration/0
```

Рисунок 26 – Нагрузка на СУБД PostgreSQL-PRO

Результат теста TPC-1C предоставляется виде диаграммы, цвет которой обозначает оценку производительности (рисунок 27).

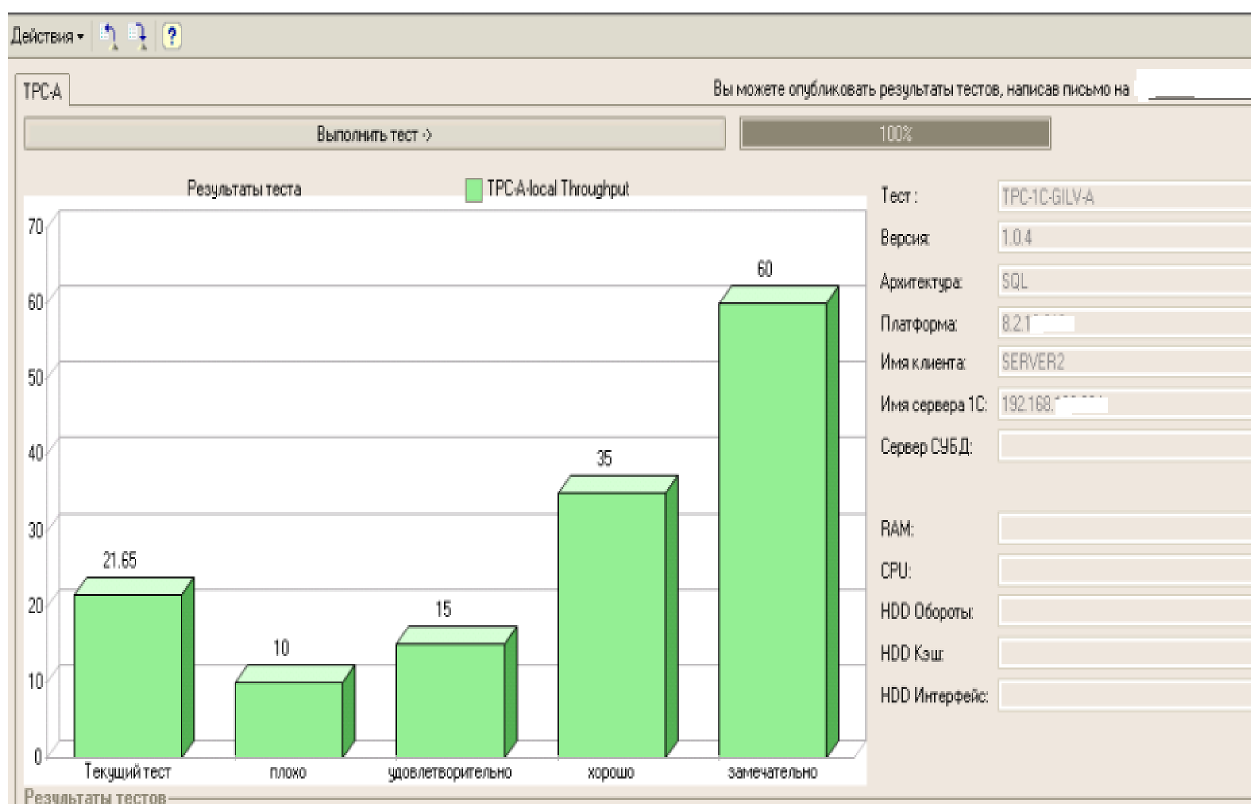

Рисунок 27 – Диаграмма результатов тестирования

На представленной диаграмме зеленый цвет дает возможность сделать общую оценку о достаточной степени производительности используемых вычислительных мощностей.

На основе представленной методики проведено функциональное тестирование при помощи специальных средств автоматизации тестирования, что способно гарантировать эффективность процесса благодаря снижению человеческого фактора.

Выводы к главе 4

Чтобы автоматизировать процесс тестирования конфигурации используются встроенные в платформу механизмы автоматизации.

Для проведения функционального тестирования бизнес- приложения разработан сценарий, которой был реализован в виде внешних обработок.

На основе авто-теста TPC-1C было произведено нагрузочное тестирование сервера приложений 1С, терминального сервера и СУБД PostgreSQL-PRO, позволяющее сделать общую оценку о допустимой производительности используемых серверов.

Основываясь на разработанной методике, с помощью специализированных средств автоматизации, было проведено функциональное тестирование бизнес-приложения, позволяющее повысить эффективность процесса благодаря минимизации влияния человеческого фактора.

Заключение

В процессе диссертационного исследования был проведен анализ проблем и методов тестирования платформенных бизнес-приложений. Проведен анализ наиболее распространенных типов нефункционального тестирования.

Рассмотрены действия тестировщика на различных этапах Scrum и проведен анализ метода автоматизированного тестирования. Выявлены преимущества и недостатки существующих методик тестирования платформенных бизнес-приложений.

В ходе исследования получены следующие основные результаты:

– Проведенный анализ подтвердил отсутствие общих рекомендаций по применению конкретных методов тестирования для тестирования платформенных бизнес-приложений.

– Был проведен анализ методов тестирования, который показал, что могут применяться разные технологии тестирования.

– Проведенный анализ имеющихся методик тестирования платформенных бизнес-приложений, подтвердил неимение многоцелевой методики тестирования, а также подтвердил актуальность темы исследования. Была разработана уникальная методика тестирования бизнес-приложений.

– Осуществлена апробация и подтверждена возможность использования предложенной методики для повышения действенности тестирования платформенных бизнес-приложений.

Благодаря выполненным исследованиям в работе решена научно-исследовательская задача разработки методики тестирования платформенных бизнес-приложений, способная повысить эффективность данного процесса.

Проведенная работа имеет большую значимость, так как благодаря использованию данной методики возможно повышение действенности тестирования платформенных бизнес-приложений.

Список используемой литературы

1. ГОСТ Р 53622-2009 Информационные технологии. Информационно- вычислительные системы. Стадии и этапы жизненного цикла, виды и комплектность документов.

2. ГОСТ Р 56922-2016. Системная и программная инженерия. Тестирование программного обеспечения.

3. Макконнелл С. Совершенный код. Мастер-класс / С. Макконнелл. –М.: Русская редакция, 2017. - 896 с.

4. Нуралиев С.Г. Архитектура «1С:Предприятия» как продукт инженерной мысли / С.Г. Нуралиев // PC Week/ Russuan Edition. -2004. - №№ 46-48.

5. Шайхутдинова А.Ф. Тестирование производительности веб-приложений: основные приемы генерации нагрузки и мониторинга // European science. 2015. №6 (7).

6. Баркалов С. А., Азарнова Т.В., Полухин П.В. Управление процессом тестирования веб-приложений методом фаззинга на основе динамических байесовских сетей // Вестник ЮУрГУ. Серия: Компьютерные технологии, управление, радиоэлектроника. 2017. №2.

7. Мартюков А. С. О необходимости разработки гибкого процесса тестирования интернет-приложений // Новые информационные технологии в автоматизированных системах. 2011. №14.

8. 1С: Предприятие 8 [Электронный ресурс]. — Режим доступа: http://v8.1c.ru/ (дата обращения: 09.03.2022).

9. Берендеев И. Программный комплекс «1С: Предприятие 8.0» как платформа разработки бизнес-приложений КТПП [Электронный ресурс] / И. Берендеев. — Режим доступа: https://sapr.ru/article/7537 (дата обращения:

09.03.2022).

10. Загрузки веб-сервисов [Электронный ресурс]. — Режим доступа:

https://infostart.ru/public/1014870/?utm_source=subscribe&utm_campaign=wee k&u tm_term=16 (дата обращения: 09.03.2022).

11. Котляров В. П. Основы тестирования программного обеспечения [Электронный ресурс] / В. П. Котляров. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 334 с. — Режим доступа: http://www.iprbookshop.ru/62820.html (дата обращения: 09.03.2022).

12. Липаев В. В. Тестирование компонентов и комплексов программ [Электронный ресурс] : учебник / В. В. Липаев. — М. : СИНТЕГ, 2010. — 393 с.

— Режим доступа: http://www.iprbookshop.ru/27301.html (дата обращения: 09.03.2022).

13. Методическая поддержка для разработчиков и администраторов 1С:Предприятия 8 [Электронный ресурс]. — Режим доступа: https://its.1c.ru/db/metod8dev/content/2290/hdoc (дата обращения: 09.03.2022).

14. Нагрузочный тест TPC-1C [Электронный ресурс]. — Режим доступа: http://www.gilev.ru/tpc1cgilv/ (дата обращения: 09.03.2022).

15. Оценка эффективности автоматизации тестирования [Электронный ресурс]. — Режим доступа: https://www.a1qa.ru/blog/otsenka-effektivnosti- avtomatizatsii-testirovaniya/ (дата обращения: 09.03.2022).

16. Савастюк С. Методологии тестирования ПО. Какую выбрать? [Электронный ресурс] / Савастюк С. — Режим доступа: https://xbsoftware.ru/blog/metodologii-testirovaniya-po-kakuyu-vybrat/ (дата обращения 09.03.2022).

17. Соловьев С.В. Технология разработки прикладного программного обеспечения / С.В. Соловьев, Р.И. Цой, Л.С. Гринкруг. –М.: Академия Естествознания, 2011. — Режим доступа: https://www.monographies.ru/ru/book/view?id=141 (дата обращения 09.03.2022).

18. AQA – система автоматизированного тестирования бизнес-приложений [Электронный ресурс]. — Режим доступа: https://www.galaktika.by/aqa-cistema-avtomatizirovannogo-testirovaniya-biznes- prilozhenij.html (дата обращения: 09.03.2022).

19. Business Application Testing [Электронный ресурс]. — Режим доступа: https://www.precisetestingsolution.com/business-application-testing.php (дата обращения: 09.03.2022).

20. Functional Testing: A Complete Guide with Types and Example [Электронный ресурс]. — Режим доступа: https://www.softwaretestinghelp.com/guide-to-functional-testing/ (дата обращения: 09.03.2022).

21. Gartner consulting [Электронный ресурс]. — Режим доступа: https://www.gartner.com (дата обращения 09.03.2022).

22. Guru99 [Электронный ресурс]. — Режим доступа: https://www.guru99.com/ (дата обращения: 09.03.2022).

23. How to Write Test Cases: Sample Template with Examples [Электронный ресурс]. — Режим доступа: https://www.guru99.com/test-case.html (дата обращения: 09.03.2022).

24. Software Testing Fundamentals [Электронный ресурс]. — Режим доступа: http://softwaretestingfundamentals.com/ (дата обращения: 09.03. 2022).

25. Testing SAP applications [Электронный ресурс]. — Режим доступа: https://www.ibm.com/support/knowledgecenter/en/SSBLQQ_9.2.1/com.ibm.rational. test.ft.doc/topics/r_taskflow_sap.html (дата обращения: 09.03.2022).

26. What Is ERP Testing and Why Does It Matter? [Электронный ресурс]. Режим доступа: https://www3.technologyevaluation.com/research/article/what- is-erp-testing-and-why-does-it-matter.html (дата обращения: 09.03.2022).

27. Ambler S.W. and Sadalage P. “Database Refactoring:

Evolutionary Database Design", Boston: Prentice Hall PTR. - 2006.

28. Li E. "Software Testing in a System Development Process: A Life Cycle Perspective", Journal of Systems Management. – 1990. - 41(8). - pp. 23-31.

29. Lutteroth C., Weber G. "Modeling a Realistic Workload for Performance Testing" in Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference Washington DC USA:IEEE Computer Society. – 2008. - pp. 149-158.

30. Monsma J.R. "Model-based testing of Web applications", Radboud University. - 2015.

31. Scott Barber R. "Load Models for Performance Testing with Incomplete Empirical Data", PerfTestPlus, Inc. -2011.

32. Zarrad A. "A systematic review on regression testing for web-based applications". – 2015. - JSW10(8):971–990.

Приложение А

Сценарии тестирования модуля справочника бизнес-приложения 1С8

```
#Если Сервер ИЛИ ТолстыйКлиентОбычноеПриложение ИЛИ ВнешнееСоединение Тогда

#Область ОписаниеПеременных

Перем ЭтоКопия Экспорт;    // Флаг копирования объекта.
Перем ОбъектКопия Экспорт; // Содержит ссылку на источник копирования.

#КонецОбласти

#Область ОбработчикиСобытий

// Обработчик ПередЗаписью
//
Процедура ПередЗаписью(Отказ)

	Если ОбменДанными.Загрузка Тогда
		Возврат;
	КонецЕсли;

	Если ЭтоГруппа Тогда
		Возврат;
	КонецЕсли;

КонецПроцедуры

Процедура ПриКопировании(ОбъектКопирования)

	ЭтоКопия = Истина;
	ОбъектКопия = ОбъектКопирования.Ссылка;

КонецПроцедуры

Процедура ПриЗаписи(Отказ)

	Если ОбменДанными.Загрузка Тогда
		Возврат;
	КонецЕсли;

	Если ЭтоКопия Тогда
		СкопироватьКонфигурацию(ОбъектКопия, Ссылка);
		ЭтоКопия = Ложь;
	КонецЕсли;
```

```
КонецПроцедуры

#КонецОбласти

#Область СлужебныеПроцедурыИФункции

Процедура СкопироватьКонфигурацию(Источник, Приемник)

	ВерсияИсточник = НайтиПоследнююВерсию(Источник, Ложь);
	Если ЗначениеЗаполнено(ВерсияИсточник) Тогда
		ВерсияПриемник =
СкопироватьВерсиюКонфигурации(Источник, Приемник, ВерсияИсточник);
		СкопироватьСтруктуруКонфигурации(ВерсияИсточник,
ВерсияПриемник);
		ТаблицаНомеровОшибок =
СкопироватьОшибкиКонфигурации(ВерсияИсточник, ВерсияПриемник);

	СкопироватьКомментарииНайденныхОшибок(ТаблицаНомеровОшибок);
	Иначе
		Сообщить(НСтр("ru='Не найдено ни одной версии
конфигурации-источника.'") + " "
			+ НСтр("ru='Версия и структура конфигурации не будут
скопированы.'"));
	КонецЕсли;

КонецПроцедуры

Функция СкопироватьОшибкиКонфигурации(ВерсияИсточник,
ВерсияПриемник)

	ТаблицаНомеровОшибок = Новый ТаблицаЗначений;
	ТаблицаНомеровОшибок.Колонки.Добавить("СтарыйНомер",
Новый ОписаниеТипов("Число"));
	ТаблицаНомеровОшибок.Колонки.Добавить("НовыйНомер", Новый
ОписаниеТипов("Число"));
	ТаблицаНомеровОшибок.Колонки.Добавить("Объект", Новый
ОписаниеТипов("СправочникСсылка.СтруктураКонфигурации"));

	Запрос = Новый Запрос;
	Запрос.Текст = "
	|ВЫБРАТЬ
	|	НайденныеОшибки.Правило,
	|	НайденныеОшибки.Номер КАК Номер,
	|	НайденныеОшибки.Ошибка,
	|	НайденныеОшибки.Состояние,
```

```
		|	НайденныеОшибки.Ответственный,
		|	НайденныеОшибки.АвторОсобенности,
		|	НайденныеОшибки.ДатаПомещенияВОсобенности,
		|	НайденныеОшибки.МестоОбнаружения,
		|	НайденныеОшибки.Уточнение,
		|	НайденныеОшибки.ДатаМодификации,
		|	НайденныеОшибки.ПричинаОсобенности,
		|	ЕСТЬNULL(СтруктураКонфигурации.Ссылка, &ПустаяСсылка) КАК Объект
		|ИЗ
		|	РегистрСведений.НайденныеОшибки КАК НайденныеОшибки
		|		ЛЕВОЕ СОЕДИНЕНИЕ Справочник.СтруктураКонфигурации КАК СтруктураКонфигурации
		|		ПО (НайденныеОшибки.Объект.Путь = СтруктураКонфигурации.Путь)
		|			И (НайденныеОшибки.Объект.ТипОбъекта = СтруктураКонфигурации.ТипОбъекта)
		|			И (СтруктураКонфигурации.Владелец = &ВерсияПриемник)
		|ГДЕ
		|	НайденныеОшибки.Объект.Владелец = &ВерсияИсточник
		|
		|УПОРЯДОЧИТЬ ПО
		|	Объект, Номер";

		Запрос.УстановитьПараметр("ВерсияИсточник", ВерсияИсточник);
		Запрос.УстановитьПараметр("ВерсияПриемник", ВерсияПриемник);
		Запрос.УстановитьПараметр("ПустаяСсылка", Справочники.СтруктураКонфигурации.ПустаяСсылка());

		Выборка = Запрос.Выполнить().Выбрать();

		ПоследнийНомерОшибки = ПолучитьМаксимальныйНомерОшибки();

		НайденныеОшибкиНаборЗаписей = РегистрыСведений.НайденныеОшибки.СоздатьНаборЗаписей();

		Пока Выборка.Следующий() Цикл

			ПоследнийНомерОшибки = ПоследнийНомерОшибки + 1;

			СтрокаТаблицы = ТаблицаНомеровОшибок.Добавить();
			СтрокаТаблицы.СтарыйНомер = Выборка.Номер;
```

```
			СтрокаТаблицы.НовыйНомер = ПоследнийНомерОшибки;
			СтрокаТаблицы.Объект = Выборка.Объект;

			НайденныеОшибкиНаборЗаписей.Отбор.Объект.Значение =
Выборка.Объект;
	НайденныеОшибкиНаборЗаписей.Отбор.Объект.Использование = Истина;
			НайденныеОшибкиНаборЗаписей.Прочитать();

			НоваяЗапись = НайденныеОшибкиНаборЗаписей.Добавить();
			ЗаполнитьЗначенияСвойств(НоваяЗапись, Выборка);
			НоваяЗапись.Номер = ПоследнийНомерОшибки;

			НайденныеОшибкиНаборЗаписей.Записать();

		КонецЦикла;

		Возврат ТаблицаНомеровОшибок;

	КонецФункции

	Функция
СкопироватьКомментарииНайденныхОшибок(ТаблицаНомеровОшибок)

		Запрос = Новый Запрос;
		Запрос.МенеджерВременныхТаблиц = Новый
МенеджерВременныхТаблиц;
		Запрос.УстановитьПараметр("ТаблицаНомеровОшибок",
ТаблицаНомеровОшибок);
		Запрос.Текст = "
		|ВЫБРАТЬ
		|	ТаблицаНомеровОшибок.Объект,
		|	ТаблицаНомеровОшибок.СтарыйНомер,
		|	ТаблицаНомеровОшибок.НовыйНомер
		|ПОМЕСТИТЬ ТаблицаНомеров
		|ИЗ
		|	&ТаблицаНомеровОшибок КАК ТаблицаНомеровОшибок
		|;
		|
		|////////////////////////////////////////////////////////////////////////////////
		|ВЫБРАТЬ
		|	КомментарииНайденныхОшибок.Комментарий,
		|	ТаблицаНомеров.Объект КАК Объект,
		|	ТаблицаНомеров.НовыйНомер КАК Номер
		|ИЗ
```

```
|	ТаблицаНомеров КАК ТаблицаНомеров
|		ВНУТРЕННЕЕ СОЕДИНЕНИЕ РегистрСведений.КомментарииНайденныхОшибок КАК КомментарииНайденныхОшибок
|		ПО ТаблицаНомеров.СтарыйНомер = КомментарииНайденныхОшибок.Номер
|
|УПОРЯДОЧИТЬ ПО
|	Объект, Номер";

	ТаблицаКомментарииНайденныхОшибок = Запрос.Выполнить().Выгрузить();

	КомментарииНайденныхОшибокНаборЗаписей = РегистрыСведений.КомментарииНайденныхОшибок.СоздатьНаборЗаписей();

	Для Каждого КомментарийНайденныхОшибок Из ТаблицаКомментарииНайденныхОшибок Цикл


КомментарииНайденныхОшибокНаборЗаписей.Отбор.Объект.Установить(КомментарийНайденныхОшибок.Объект);
		КомментарииНайденныхОшибокНаборЗаписей.Прочитать();

		НоваяЗапись = КомментарииНайденныхОшибокНаборЗаписей.Добавить();
		ЗаполнитьЗначенияСвойств(НоваяЗапись, КомментарийНайденныхОшибок);

		КомментарииНайденныхОшибокНаборЗаписей.Записать();

	КонецЦикла;

КонецФункции

Функция СкопироватьВерсиюКонфигурации(Источник, Приемник, ВерсияИсточник)

	МассивСвойствДляИсключения = Новый Массив;
	МассивСвойствДляИсключения.Добавить("Владелец");
	МассивСвойствДляИсключения.Добавить("Родитель");
	МассивСвойствДляИсключения.Добавить("СобранныеДанные");

	СвойстваДляИсключения = СтрСоединить(МассивСвойствДляИсключения, ", ");
```

Продолжение Приложения А

```
		НовыйЭлемент = Справочники.Версии.СоздатьЭлемент();
		НовыйЭлемент.Владелец = Приемник;
		ЗаполнитьЗначенияСвойств(НовыйЭлемент, ВерсияИсточник,, СвойстваДляИсключения);
		НовыйЭлемент.Записать();

		Возврат НовыйЭлемент.Ссылка;

	КонецФункции

	Процедура СкопироватьПодсистемыОбъекта(ОбъектИсточник, ОбъектПриемник)

		Для Каждого ПодсистемаИсточник Из ОбъектИсточник.Подсистемы Цикл

			ПодсистемаПуть = ПодсистемаИсточник.Подсистема.Путь;
			ВерсияПриемник = ОбъектПриемник.Владелец;
			ПодсистемаСсылка = Справочники.СтруктураКонфигурации.НайтиПоРеквизиту("Путь", ПодсистемаПуть,, ВерсияПриемник);

			ПодсистемаПриемник = ОбъектПриемник.Подсистемы.Добавить();
			ПодсистемаПриемник.Подсистема = ПодсистемаСсылка;

		КонецЦикла;

	КонецПроцедуры

	Процедура СкопироватьСтруктуруКонфигурации(ВерсияИсточник, ВерсияПриемник)

		ЗапросПоОбъектам = Новый Запрос;
		ЗапросПоОбъектам.Текст = "
		|ВЫБРАТЬ
		|	СтруктураКонфигурации.Ссылка
		|ИЗ
		|	Справочник.СтруктураКонфигурации КАК СтруктураКонфигурации
		|ГДЕ
		|	СтруктураКонфигурации.Владелец = &Владелец
		|	И НЕ СтруктураКонфигурации.ПометкаУдаления
		|
```

```
		|УПОРЯДОЧИТЬ ПО
		|	СтруктураКонфигурации.НомерПоПорядку";

		ЗапросПоОбъектам.УстановитьПараметр("Владелец",
ВерсияИсточник);

		Выборка = ЗапросПоОбъектам.Выполнить().Выбрать();
		НомерОбъекта = 0;
		ВсегоОбъектов = Выборка.Количество();
		Пока Выборка.Следующий() Цикл

			СсылкаИсточник = Выборка.Ссылка;
			ОбъектПриемник =
Справочники.СтруктураКонфигурации.СоздатьЭлемент();

			ЗаполнитьЗначенияСвойств(ОбъектПриемник,
СсылкаИсточник,, "Владелец, Родитель, Код");

			РодительПуть = Выборка.Ссылка.Родитель.Путь;
			РодительСсылка =
Справочники.СтруктураКонфигурации.НайтиПоРеквизиту("Путь",
РодительПуть,, ВерсияПриемник);
			ОбъектПриемник.Родитель = РодительСсылка;
			ОбъектПриемник.Владелец = ВерсияПриемник;

			СкопироватьПодсистемыОбъекта(СсылкаИсточник,
ОбъектПриемник);

			ЭлементЗаписан = Ложь;
			СчетчикТранзакций = 1;

			Пока (НЕ ЭлементЗаписан) И (СчетчикТранзакций < 1000)
Цикл
				ЭлементЗаписан = Истина;
				Попытка
					ОбъектПриемник.Записать();
				Исключение
					ЭлементЗаписан = Ложь;
				КонецПопытки;
				СчетчикТранзакций = СчетчикТранзакций + 1;
			КонецЦикла;

			НомерОбъекта = НомерОбъекта + 1;
			#Если Клиент Тогда
			ТекстСостояния = НСтр("ru='Выполняется запись структуры
```

```
конфигурации.'") + " "
                        + СтрШаблон(НСтр("ru='Объект конфигурации №%1 из
%2: %3.'"), НомерОбъекта, ВсегоОбъектов, ОбъектПриемник.Путь);
                  Состояние(ТекстСостояния);
                  #КонецЕсли

            КонецЦикла;

      КонецПроцедуры

      #КонецОбласти

      #Область Инициализация

      ЭтоКопия = Ложь;

      #КонецОбласти

      #КонецЕсли
```