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

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

Кафедра «Прикладная математика и информатика»
(наименование кафедры)

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

Информационные системы и технологии корпоративного управления
(направленность (профиль)/специализация

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

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

Студент Р.Е. Семенчук _____
(И.О. Фамилия) (личная подпись)

Научный руководитель О.М. Гущина _____
(И.О. Фамилия) (личная подпись)

Руководитель программы д.т.н., доцент, С.В. Мкртычев _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____»_____20_____г.

Допустить к защите
Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____
«_____» _____20_____г. (личная подпись)

Тольятти 2019

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Одним из перспективных направлений в области управления производственно-хозяйственной деятельности современных предприятий и компаний является активная интеграция в их ИТ-инфраструктуру высокотехнологичных бизнес-приложений.

Под бизнес-приложением понимается любое программное обеспечение или программный комплекс, используемые для выполнения различных бизнес-задач, в том числе для повышения производительности и эффективности предприятий и компаний (ERP-системы, CRM-системы и др.).

Новые технологии и тенденции создают новые возможности для бизнеса-интеграции внутренних операций, отношений с клиентами и отраслевых производственно-сбытовых цепочек.

Как показывает практика, наиболее востребованными в настоящее время являются бизнес-приложения, реализованные на основе современных технологических платформ (1С: Предприятие 8.х, Галактика, SAP и др.).

Для обеспечения высокого качества бизнес-приложений необходимо выполнить их тестирование в процессе проектирования.

Вполне понятно, что развитие современной компании-разработчика программного обеспечения невозможно без внедрения новых методов, моделей и инструментов тестирования программного обеспечения, обеспечивающих проверку качества и функциональности платформенных бизнес-приложений.

Таким образом, **актуальность магистерской работы** обусловлена необходимостью исследования и разработки методики тестирования, обеспечивающей повышение эффективности процесса тестирования платформенных бизнес-приложений.

Объектом исследования является процесс тестирования бизнес-приложений.

Предметом исследования является методика тестирования платформенных бизнес-приложений.

Целью работы является исследование и разработка методики тестирования платформенных бизнес-приложений, обеспечивающей повышение эффективности данного процесса.

Гипотеза исследования: применение предлагаемой методики позволит повысить эффективность процесса тестирования платформенных бизнес-приложений.

Для достижения цели и проверки сформулированной гипотезы необходимо решить следующие задачи:

1. Проанализировать существующие виды тестирования программного обеспечения и оценить возможность их применения для платформенных бизнес-приложений.
2. Проанализировать методы и модели тестирования программного обеспечения и оценить возможность их применения для платформенных бизнес-приложений.
3. Разработать методику тестирования платформенных бизнес-приложений.
4. Выполнить апробацию и обосновать применение предложенной методики для повышения эффективности тестирования платформенных бизнес-приложений.

Новизна исследования заключается в разработке методики тестирования платформенных бизнес-приложений.

На защиту выносятся:

1. Методика тестирования платформенных бизнес-приложений.
2. Результаты апробации методики тестирования платформенных бизнес-приложений.

Практическая значимость магистерской работы заключается в реализации методики тестирования, обеспечивающей надежную проверку платформенных бизнес-приложений в компаниях-разработчиках программного обеспечения на платформе «1С: Предприятие 8».

Методы исследования: системный анализ, методы тестирования программного обеспечения, методологии проектирования бизнес-приложений.

Публикации по теме исследования. Результаты исследования представлены в статье: *Семенчук Р.Е. Методика тестирования платформенных бизнес-приложений / Р.Е. Семенчук // Вестник научных конференций. -2019. - № 4.*

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

Первая глава посвящена анализу видов тестирования бизнес-приложений. Рассмотрены особенности функциональных и нефункциональных видов тестирования программного обеспечения.

Во второй главе проанализированы методы и модели тестирования бизнес-приложений. Описаны достоинства и недостатки методов и методологий тестирования на основе жизненного цикла бизнес-приложений.

Третья глава посвящена разработке методики тестирования бизнес-приложений, реализованных на платформе «1С: Предприятие 8».

В четвертой главе представлены результаты апробации предлагаемой методики тестирования бизнес-приложений.

В заключении подводятся итоги выполненной работы.

В приложении представлен код сценария модульного тестирования бизнес-приложения.

Работа изложена на 81 с. и включает рисунков 33, таблиц 4.

Глава 1 АНАЛИЗ ВИДОВ ТЕСТИРОВАНИЯ БИЗНЕС-ПРИЛОЖЕНИЙ

1.1 Основные определения и термины

Платформенное бизнес-приложение – бизнес-приложение, разработанное на базе программной технологической платформы, которая представляет собой среду исполнения и набор технологий, используемые в качестве основы для разработки специализированного программного обеспечения (1С8, Галактика, SAP и др.) [18].

К платформенным бизнес-приложениям предъявляются требования, характерные для всех видов бизнес-приложений:

- легкость адаптации к потребностям конкретной компании;
- легкость внедрения и освоения персоналом;
- легкость добавления нового функционала.

Тестирование программного обеспечения - это вид деятельности в области разработки программного обеспечения. Это исследование, проводимое в отношении программного обеспечения для предоставления заинтересованным сторонам информации о его качестве.

Тестирование программного обеспечения связано с понятиями верификации и валидации.

Верификация - это проверка или тестирование элементов, включая программное обеспечение, на соответствие спецификации.

Валидация - это процесс проверки того, что спецификация программного обеспечения соответствует требованиям пользователя.

Тест-кейс - набор действий, выполняемых для проверки конкретной функции или функциональности программного приложения. Тест-кейс является обязательным компонентом жизненного цикла тестирования программного обеспечения [20].

Анализ работ по данной тематике показал, что для тестирования бизнес-приложений используются различные виды функционального и нефункционального тестирования [15, 16].

Рассмотрим особенности каждого из данных видов тестирования.

1.2 Функциональное тестирование бизнес-приложений

Функциональное тестирование - это тестирование приложения на соответствие бизнес-требованиям [9, 17].

Оно включает в себя все типы тестов, разработанные для гарантии того, что каждая часть программного обеспечения (ПО) ведет себя так, как ожидается, используя кейсы, предоставленные командой разработчиков или бизнес-аналитиком.

На рисунке 1.1 изображена структурная схема процесса функционального тестирования ПО.

Рисунок 1.1 – Структурная схема процесса функционального тестирования ПО

Функциональное тестирование включает в себя следующие уровни тестирования:

- модульное (юнит) тестирование;
- интеграционное тестирование;

− системное тестирование;

− приемочное тестирование.

Рассмотрим каждый из представленных уровней функционального тестирования.

1.2.1 Модульное тестирование бизнес-приложений

Модульное или юнит-тестирование является первым уровнем тестирования и часто выполняется самими разработчиками.

Это процесс подтверждения того, чтобы отдельные компоненты программного приложения на уровне кода работают в соответствии с техническим заданием на его разработку.

Разработчики в среде, управляемой тестами, обычно пишут и запускают тесты до того, как ПО будет передано группе тестирования.

Модульное тестирование облегчает отладку, потому что при обнаружении проблем на ранней стадии для их устранения требуется меньше времени, чем при обнаружении на более поздних стадиях тестирования приложения.

Практически модульное тестирование заключается в написании фрагмента кода (модульного теста) для проверки кода (модуля), написанного для реализации требований.

На рисунке 1.2 представлен цикл модульного тестирования.

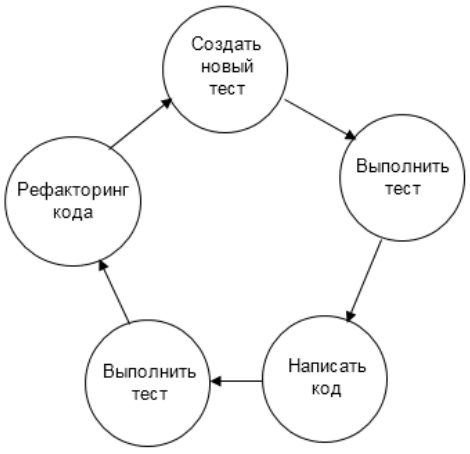

Рисунок 1.2 – Цикл модульного тестирования ПО

Достоинства модульного тестирования:

− тестирование можно проводить на ранних этапах жизненного цикла разработки ПО, когда другие модули могут быть недоступны для интеграции;

− исправление проблемы в модульном тестировании может предотвратить проблемы, возникающие на более поздних этапах разработки и тестирования;

− стоимость исправления дефекта, обнаруженного в модульном тестировании, значительно ниже стоимости, обнаруженной в системе или приемочных испытаниях;

− полнота кода может быть продемонстрирована с помощью модульных тестов. Это более полезно в гибком процессе. Тестировщики не получат функциональные сборки для тестирования, пока интеграция не будет завершена;

− повышение надежности проектирования и разработки, поскольку разработчики пишут тестовые примеры, сначала разбираясь в спецификациях.

− позволяет просто определить виновника ошибки;

− экономит времени разработки: завершение кода может занять больше времени, но из-за уменьшения количества дефектов можно сэкономить общее время разработки.

Несмотря на то, что модульное тестирование может проводиться вручную, автоматизация данного процесса ускорит циклы доставки и расширит охват тестирования.

1.2.2 Интеграционное тестирование бизнес-приложений

Интегрированные тесты могут проводиться разработчиками или независимыми тестировщиками и обычно состоят из комбинации автоматизированных функциональных и ручных тестов.

Основная функция или цель этого тестирования - проверить интерфейсы между модулями.

Отдельные модули сначала тестируются изолированно. Сразу после того, как модули проходят модульное тестирование, они интегрируются друг с другом, чтобы проверить их комбинационное поведение и правильность реализации требований к ним или нет.

Наиболее распространены следующие подходы к интеграционному тестированию:

1) Подход «большого взрыва».

Подход «большого взрыва» основан на одновременной интеграции всех модулей. Проверяется, работает ли система, как ожидалось, или нет, после интеграции (рисунок 1.3). Вместе с тем, если какая-либо проблема обнаружена в полностью интегрированном модуле, становится трудно определить, какой модуль вызвал данную проблему.

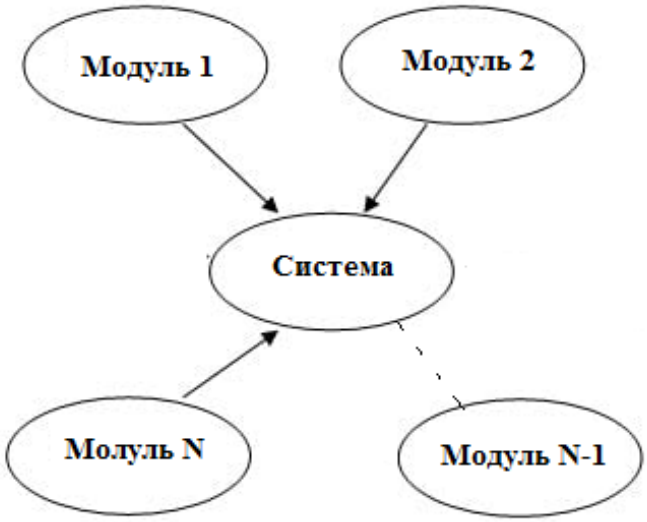

Рисунок 1.3 – Модель интеграционного тестирования по методу «большого взрыва»

Подход «большого взрыва» - это трудоемкий процесс поиска модуля, у которого есть дефект, так как для этого потребуется время, и как только дефект будет обнаружен, его устранение будет стоить дорого, поскольку дефект будет обнаружен на более поздней стадии.

2) Восходящее тестирование.

Восходящее тестирование, как следует из названия, начинается с самого нижнего или самого внутреннего блока приложения и постепенно перемещается вверх.

Соответственно интеграционное тестирование начинается с самого низкого модуля и постепенно переходит к верхним модулям приложения. Эта интеграция продолжается до тех пор, пока все модули не будут интегрированы и все приложение не будет протестировано как единое целое.

На рисунке 1.4 модули B1C1, B1C2 и B2C1, B2C2 являются модулями самого низкого уровня, которые проходят модульное тестирование. Модули B1 и B2 еще не разработаны. Функциональность модулей B1 и B2 заключается в том, что он вызывает модули B1C1, B1C2 и B2C1, B2C2.

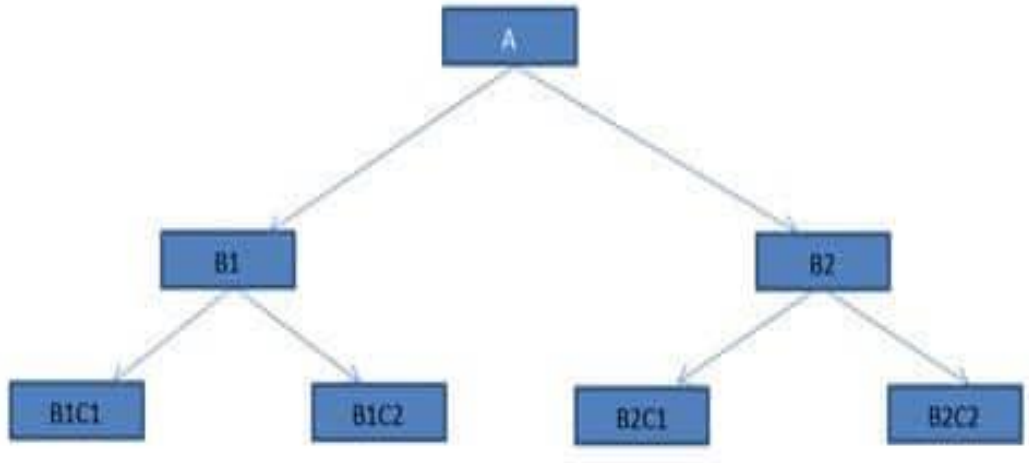

Рисунок 1.4 – Диаграмма иерархии модулей ПО

Поскольку B1 и B2 еще не разработаны, понадобится какая-то программа или стимулятор, который будет вызывать модули B1C1, B1C2 и B2C1, B2C2.

Эти программы-стимуляторы называются драйверами.

Иными словами, драйвер - это фиктивная программа, которая используется для вызова функций самого нижнего модуля в случае, когда вызывающая функция не существует.

Восходящий метод требует, чтобы драйвер модуля передавал входные данные тестового примера в интерфейс тестируемого модуля.

Преимущество этого подхода заключается в том, что, если в самом нижнем модуле программы существует серьезная ошибка, ее легче обнаружить и предпринять корректирующие меры.

Недостатком данного метода является то, что основная программа фактически не существует до тех пор, пока последний модуль не будет интегрирован и протестирован. В результате недостатки проектирования ПО более высокого уровня будут обнаружены только в конце процесса.

3) Нисходящее тестирование.

Процесс нисходящего тестирования начинается с самого верхнего модуля и постепенно продвигается к нижним модулям. Только верхний модуль тестируется отдельно. После этого нижние модули интегрируются один за другим. Процесс повторяется до тех пор, пока все модули не будут интегрированы и протестированы.

На рисунке 1.4 тестирование начинается с модуля A, и нижние модули B1 и B2 интегрируются один за другим. Теперь здесь нижние модули B1 и B2 фактически не доступны для интеграции. Поэтому, чтобы протестировать самый верхний модуль A, разрабатывается т.н. «заглушки».

Заглушка - это фрагмент кода, который принимает входные данные / запросы от верхнего модуля и возвращает результаты / ответ. Таким образом, несмотря на отсутствие нижних модулей, существует возможность тестирования верхнего модуля.

В практических сценариях поведение заглушек не так просто, как кажется. Современные программные модули в основном используют сложную бизнес-логику, такую, например, как подключение к базе данных. В результате создание заглушек становится таким же сложным и отнимает много времени, как и создание реального модуля.

Следует учесть, что и заглушки, и драйверы являются фиктивным фрагментом кода, который используется для тестирования «несуществующих» модулей. Они запускают функции / метод и возвращают ответ, который сравнивается с ожидаемым поведением.

Необходимо также иметь в виду, что интеграционное тестирование не происходит в конце цикла, а проводится одновременно с разработкой. Так что в большинстве случаев все модули на самом деле не доступны для тестирования, и зачастую приходится проверять несуществующие программные компоненты.

1.2.3 Системное тестирование бизнес-приложений

Системное тестирование представляет собой тестирование программной системы в целом. Все модули / компоненты объединяются, чтобы проверить, работает ли система как ожидалось или нет.

Системное тестирование проводится после интеграционного тестирования. Это играет важную роль в предоставлении высококачественного программного продукта.

Системное тестирование - это тестирование по методу «черного ящика», используемое для проверки завершенной и интегрированной системы в целом на предмет соответствия установленным требованиям (рисунок 1.5).

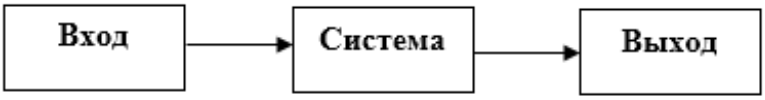

Рисунок 1.5 – Модель процесса системного тестирования

Тестирование по методу «черного ящика» - это вид тестирования ПО, при котором тестировщики не знают внутренней структуры, дизайна и кода ПО [8].

Единственной целью в ходе такого тестирования является тестирование функциональности ПО и проверка его соответствия указанным требованиям.

Именно по этой причине указанный метод иногда называют тестированием на основе спецификаций.

Иными словами, тестирование по методу «черного ящика» используется для проверки правильности и бесперебойности функционирования программной системы без анализа и оценки ее внутренней структуры.

Преимущества метода:

− обеспечивает плавную реализацию приложения в исходной системной среде;

− идеально подходит для функционального тестирования;

− полное знание функциональных спецификаций позволяет довольно быстро проектировать тест-кейсы;

− не требует знания языков программирования и методов системной реализации;

− обеспечивает объективность и непредвзятость тестирования, так как разработчик и тестировщики работают независимо друг от друга;

− может быть выполнено конечными пользователями.

Недостатки метода:

− работа с огромным пробным пространством тестовых входов может быть трудоемкой;

− вероятность встретить тупик на неопределенном пути во время тестирования довольно высока;

− тестируется только небольшая часть ПО, так как более тщательное тестирование может занять много времени;

− сложно проектировать тест-кейсы без четко определенных, точных и кратких спецификаций;

− не рекомендуется для тестирования сложных кодов и программ.

Следует также отметить, что функциональность ПО тестируется от начала до конца и, как правило, проводится отдельной группой тестировщиков, а не группой разработчиков, прежде чем продукт будет запущен в производство.

1.2.4 Приемочное тестирование бизнес-приложений

Приемочное тестирование является последним этапом функционального тестирования и используется для оценки готовности окончательного варианта ПО.

При этом имеется в виду, что прошедший приемочное тестирование программный продукт соответствует всем первоначальным бизнес-критериям и отвечает потребностям конечного пользователя.

Для этого требуется, чтобы продукт был протестирован как внутри компании, так и за ее пределами.

В этой связи при приемочном тестировании пользователей иногда приходится привлекать различных клиентов в разных странах в зависимости от продукта с предложением выполнить бета-тестирование нового ПО.

Бета-тестирование является ключом к получению реальной обратной связи от потенциальных клиентов и поможет решить любые проблемы с юзабилити ПО.

В разработке ПО бета-тестирование - это вторая фаза тестирования ПО, в которой выборочная группа пользователей из целевой аудитории испытывает продукт.

Бета-тестирование также иногда называют пользовательским приемочным тестированием или тестированием конечного пользователя.

Замечания и рекомендации пользователей передаются разработчикам, которые вносят окончательные изменения, прежде чем выпускать программный продукт на коммерческой основе.

В настоящее время ПО компании-вендоры обеспечивают доступность загрузки его бета-версии со своего интернет-портала.

Следует отметить, что у специалистов нет однозначного мнения о целесообразности применения тех или иных видов тестирования для бизнес-приложений.

Для решения данной проблемы рассмотрим и проанализируем наиболее важные специализированные виды тестирования бизнес-приложений.

1.2.5 Регрессионное тестирование бизнес-приложений

Регрессионное тестирование относится к дополнительным методам функционального тестирования.

Регрессионное тестирование используется для подтверждения того, что недавнее изменение программы или ее кода не оказало неблагоприятного воздействия на существующие функции [29].

Регрессионное тестирование - это не что иное, как полный или частичный выбор уже выполненных тест-кейсов, которые повторно выполняются, чтобы гарантировать, что существующие функции работают нормально.

Иными словами, данный метод тестирования проводится для того, чтобы убедиться, что новые изменения кода не будут иметь побочных эффектов на существующие функциональные возможности. Это гарантирует, что старый код все еще работает, как только новые изменения кода сделаны.

Регрессионное тестирование рекомендуется в следующих случаях:

- при изменении требований к ПО, приводящих к изменению кода;
- при добавлении новой функций в ПО;
- при устранении программных дефектов;
- при устранении проблем с производительностью ПО.

Классификация методов регрессионного тестирования ПО представлена на рисунке 1.6.

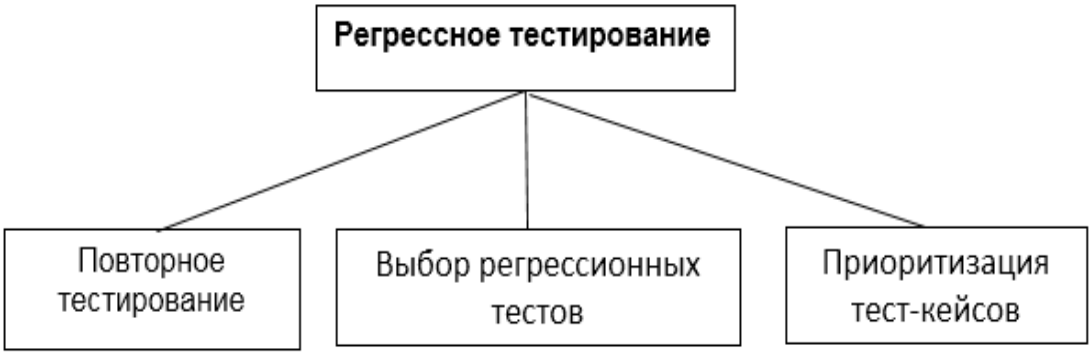

Рисунок 1.6 – Виды регрессионного тестирования ПО

Рассмотрим отдельные виды регрессионного тестирования.

1) Повторное тестирование.

Это один из методов регрессионного тестирования, при котором все тесты в существующем сегменте или наборе тестов должны быть повторно

выполнены. Данный метод связан с большими затратами, так как требует огромного времени и ресурсов.

2) Приоритизация тест-кейсов.

Заключается в расстановке приоритетов тест-кейсов в зависимости от влияния на бизнес, критических и часто используемых функций. Выбор тестовых случаев на основе приоритета значительно сократит набор регрессионных тестов.

3) Выбор регрессионных тестов.

Вместо повторного выполнения всего набора тестов, лучше выбрать готовый набор тест-кейсов (тестовый план) для выполнения.

Выбранные тестовые наборы подразделяются на:

− повторно используемые, которые могут применяться в последующих регрессионных циклах;

− устаревшие, которые не могут быть использованы в последующих циклах.

Как показывает практика, значительное количество дефектов, о которых сообщают клиенты, связано с исправлением ошибок в последнюю минуту, что приводит к побочным эффектам.

Таким образом, выбор тестового примера для регрессионного тестирования требует умения и опыта.

Ниже приведены примеры эффективных регрессионные тестов:

− тест-кейсы для частых дефектов;

− тест-кейсы, которые проверяют основные характеристики продукта;

− тест-кейсы функциональных возможностей, которые претерпели недавние изменения;

− все интеграционные тесты;

− все сложные тест-кейсы и др.

По мнению специалистов, данный метод является наиболее эффективным для тестирования бизнес-приложений.

1.3 Нефункциональное тестирование бизнес-приложений

1.3.1 Нагрузочное тестирование бизнес-приложений

Нагрузочное тестирование относится к методам нефункционального тестирования.

Нагрузочное тестирование является разновидностью тестирования производительности, которое определяет производительность системы в реальных условиях нагрузки.

В рассматриваемом контексте это тестирование помогает определить, как ведет себя бизнес-приложение, когда к нему одновременно обращаются несколько пользователей.

Нагрузочное тестирование позволяет определить:

− максимальную производительность бизнес-приложения;

− достаточность ресурсов ИТ-инфраструктуры для запуска бизнес-приложения;

− устойчивость бизнес-приложения по отношению к пиковой пользовательской нагрузке;

− количество одновременных работающих пользователей, которое может поддерживать бизнес-приложение;

− возможности масштабирования бизнес-приложения для увеличения количества одновременных работающих пользователей.

Достоинства нагрузочного тестирования:

− позволяет обнаружить узкие места бизнес-приложения или веб-сайта перед развертыванием и устранить их до того, как они повлекут за собой более реальные затраты;

− позволяет повысить масштабируемость системы и оценить потребности ИТ-инфраструктуры в ресурсах;

− обеспечивает снижение риска простоя системы, путем поиска сценариев, которые могут привести к ее сбою системы. Это делает нагрузочное тестирование отличным инструментом для поиска решений проблем с высоким трафиком до того, как они возникнут в реальных условиях;

− повышает лояльность клиентов за счет снижения времени отклика веб-веб-приложений;

− обеспечивает снижение стоимости отказов. Выявление проблем на самой ранней стадии, особенно перед запуском, снижает стоимость отказов. Напротив, сбои после запуска могут повлечь за собой экспоненциально большие затраты.

В программной инженерии нагрузочное тестирование обычно используется для тестирования клиент-серверных бизнес-приложений и веб-приложений [28].

Наиболее распространенные примеры нагрузочного тестирования:

− загрузка серии больших файлов из Интернета;

− запуск нескольких приложений на компьютере или сервере одновременно;

− назначение множества заданий принтеру в очереди;

− увеличение объема почтового трафика на сервере;

− непрерывная запись и чтение данных с жесткого диска.

Нагрузочное тестирование может проводиться двумя способами:

1) тестирование долговечности, также называемое тестированием на выносливость.

Позволяет оценить способность системы выдерживать постоянную, умеренную рабочую нагрузку в течение длительного времени.

2) объемное тестирование, подвергающее систему большой рабочей нагрузке в течение ограниченного времени.

Любой подход позволяет точно определить узкие места, ошибки и ограничения компонентов ПО.

Нагрузочное тестирование может дать пользователю общее представление о том, сколько приложений или процессов может быть запущено одновременно при сохранении номинального уровня производительности системы.

1.3.2 Стрессовое тестирование бизнес-приложений

Стрессовое тестирование – это метод тестирования ПО, которое проверяет стабильность и надежность системы.

Этот метод в основном проверяет систему на устойчивость и обработку ошибок в условиях чрезвычайно высокой нагрузки.

Иными словами, этот метод тестирует и оценивает работу ПО в экстремальных условиях.

Стрессовое тестирование проводится, чтобы убедиться, что система не будет аварийно завершать работу в критических ситуациях.

В программной инженерии стрессовое тестирование также известно как тестирование ПО на выносливость.

Ниже описаны виды стрессового тестирования.

1) Распределенное стрессовое тестирование (рисунок 1.7).

Роль стресс-сервера заключается в распространении набора стресс-тестов среди всех стресс-клиентов и отслеживании статуса клиента. После того, как клиент связывается с сервером, сервер добавляет имя клиента и начинает отправку данных для тестирования.

Так, на рисунке 1.7 сервер может соединиться с клиентами 1 и 2, но он не может отправлять или получать сигнал от клиентов 3 и 4.

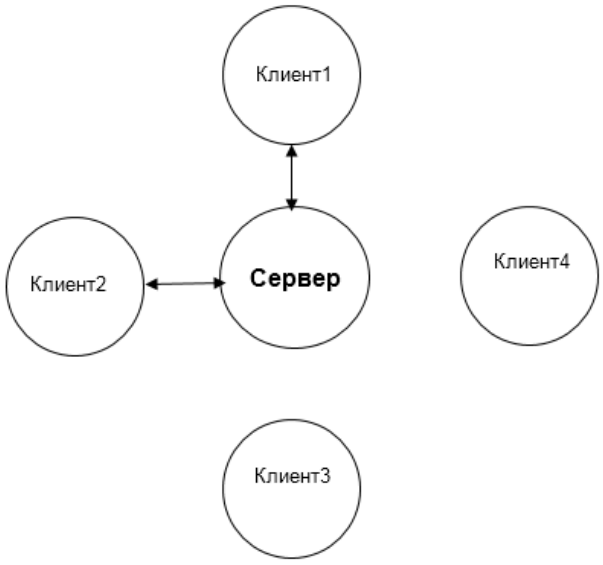

Рисунок 1.7 – Модель распределенного стрессового тестирования

Клиент посылает сообщение, что он связан с сервером. Если сервер не получает никаких сообщений от клиентского компьютера, необходимо выполнить его отладку.

2) Стресс-тестирование приложения.

Этот вид тестирования сосредоточен на обнаружении дефектов, связанных с блокировкой данных, проблемами сети и узкими местами производительности в приложении.

3)Транзакционное стресс-тестирование.

Проводится стресс-тестирование одной или нескольких транзакций между двумя или более приложениями. Данный вид стресс-тестирования используется для тонкой настройки и оптимизации системы.

3) Систематическое стресс-тестирование.

Это комплексное стресс-тестирование, которое можно проводить на нескольких системах, работающих на одном сервере.

Оно используется для поиска дефектов, когда данные одного приложения блокируют другое приложение.

4) Поисковое стресс-тестирование.

Это один из видов стресс-тестирования, который используется для тестирования системы с необычными параметрами или условиями, которые вряд ли произойдут в реальном сценарии.

Оно используется для поиска дефектов в неожиданных сценариях, таких как:

- большое количество пользователей вошли в систему одновременно;
- антивирусный сканер запущен на всех машинах одновременно;
- база данных отключилась, когда к ней обращались с веб-сайта;
- запись большого объема данных в базу данных системы.

Процесс стресс-тестирования состоит из следующих этапов:

1) планирование стресс-теста (сбор системных данных, анализ системы, определение целей стресс-теста);

2) создание сценариев автоматизации. На данном этапе создаются сценарии автоматизации стресс-тестирования, генерируются тестовые данные для стресс-сценариев;

3) выполнение скрипта. На данном этапе запускаются скрипты автоматизации стресс-тестирования и сохраняется результаты стресса;

4) анализ результатов теста и выявление узких мест;

5) настройка и оптимизация системы.

Данная процедура выполняется в режиме итерации.

Как показывает практика, для достижения целей производительности бизнес-приложений требуется от 3 до 4 циклов стресс-тестирования.

Выводы к главе 1

1) Для тестирования бизнес-приложений могут использоваться функциональное и нефункциональное тестирование. Функциональное тестирование необходимо для проверки реализуемости функциональных требований, предъявляемых к бизнес-приложениям заказчиком, и прежде всего – к поддержке ключевых бизнес-функций и процессов. Нефункциональное тестирование необходимо для проверки качественных характеристик бизнес-приложения, например, таких, как поддержка многопользовательского режима работы и юзабилити.

2) Функциональное тестирование бизнес-приложений состоит из следующих уровней тестирования: модульное (юнит) тестирование, интеграционное тестирование, системное тестирование и приемочное тестирование.

3) К нефункциональным видам тестирования бизнес-приложений относятся нагрузочное и стрессовое тестирование.

3) 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 Методы тестирования бизнес-приложений

2.1.1 Метод черного ящика

Метод черного ящика, также известный как поведенческое тестирование является методом тестирования ПО, в котором внутренняя структура, дизайн и реализация тестируемого ПО не известны тестировщику [21].

Эти тесты могут быть как функциональными (предпочтительно), так не нефункциональными.

На рисунке 2.1 представлена модель тестирования по методу черного ящика.

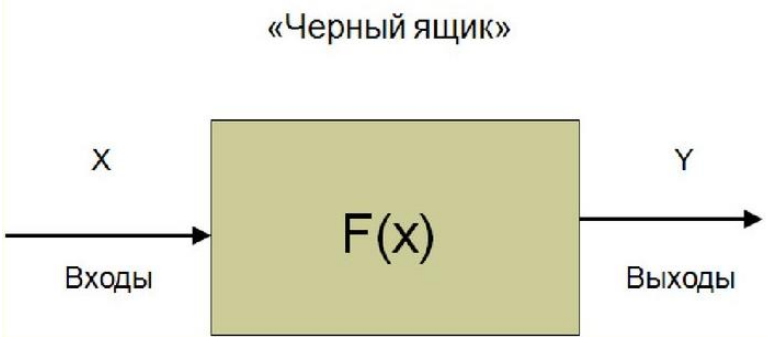

Рисунок 2.1- Модель тестирования по методу черного ящика

Данный метод позволяет найти ошибки в следующих случаях:

- неверные или отсутствующие функции;
- ошибки интерфейса;
- ошибки в структурах данных или в организации доступа к внешней базе данных;
- поведенческие ошибки или ошибки производительности;
- ошибки инициализации и завершения.

Методика тестирования по методу черного ящика представляет собой описание процедуры получения и/или выбора тест-кейсов на основе анализа спецификации, функционального или нефункционального описания компонента или системы без ссылки на его внутреннюю структуру.

Метод черного ящика применяется на следующих уровнях функционального тестирования:

− интеграционное тестирование;

− системное тестирование;

− приемочное тестирование.

Чем выше уровень, и, следовательно, чем больше и сложнее блок, тем чаще используется метод тестирования черного ящика.

Ниже приведены примеры методик тестирования по методу черного ящика:

− разделение эквивалентности: это метод проектирования программного тестирования, который включает в себя разделение входных значений на допустимые и недействительные разделы и выбор репрезентативных значений из каждого раздела в качестве тестовых данных.

− анализ граничных значений: это метод проектирования программного тестирования, который включает в себя определение границ для входных значений и выбор значений, которые находятся на границах и только внутри / снаружи границ в качестве тестовых данных;

− диаграмма причинно-следственных связей. Это методика разработки программных тестов, которая включает в себя идентификацию случаев (входных условий) и эффектов (выходных условий), создание графика причинно-следственных связей и, соответственно, создание тест-кейсов.

Преимущества метода:

− тесты выполняются с точки зрения пользователя и помогут выявить расхождения в спецификациях;

− тестировщику не нужно знать языки программирования и принципы реализации ПО;

− тесты могут проводиться независимым от разработчиков органом, что позволяет объективно оценивать ситуацию и избегать предвзятости разработчиков;

− тест-кейсы могут быть разработаны, как только спецификации будут выполнены.

Недостатки данного метода:

− можно проверить только небольшое количество возможных входных данных, при этом часть входов к ПО останутся непроверенными;

− без четких спецификаций, как это происходит во многих проектах, тест- кейсы будет сложно спроектировать;

− тесты могут быть избыточными, если разработчик ПО уже выполнил тест-кейс на своем уровне.

Таким образом, применение метода черного ящика для тестирования бизнес-приложений должно быть обосновано с учетом возможных рисков.

При тестировании бизнес-приложения методом черного ящика необходимо обеспечить участие реальных пользователей и экспертов, знающих особенности реализуемых бизнес-функций. При этом также необходимо обеспечить корректность входных данных, в качестве которых могут использоваться, например, реальные исторические данные, предоставленные пользователем.

2.1.2 Метод белого ящика

Тестирование по методу белого ящика (также известное как прозрачное тестирование, тестирование на основе кода или структурное тестирование) - это метод тестирования ПО, в котором внутренняя структура, дизайн и реализация тестируемого элемента известны тестировщику.

Тестировщик «пропускает» входные данные через код ПО и определяет соответствующие выходы (рисунок 2.2).

Знания языка программирования и особенностей реализация ПО крайне важны для тестировщика.

Тестирование белого ящика - это тестирование, выходящее за рамки пользовательского интерфейса и в мельчайших подробностях системы.

Методика тестирования по методу белого ящика основана на описании процедуры разработки и/или выбора тест-кейсов по результатам анализа внутренней структуры компонента или системы.

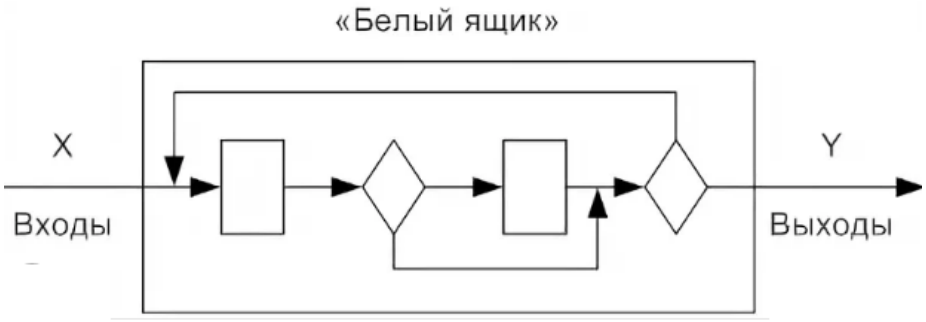

Рисунок 2.2 - Модель тестирования по методу белого ящика

Данный метод применяется на следующих уровнях тестирования ПО:

− модульное тестирование - для тестирования процессов внутри модуля;

− интеграционное тестирование - для тестирования связей между модулями;

− системное тестирование - для тестирования связей между подсистемами.

Преимущества метода:

− тестирование может быть начато на более ранней стадии при отсутствии графический интерфейса;

− тестирование более тщательное, с возможностью охвата большинства функций.

Недостатки метода:

− поскольку тесты могут быть очень сложными, требуются высококвалифицированные тестировщики с глубокими знаниями в области программирования и реализации;

− обслуживание тестового сценария может быть затратным, если реализация меняется слишком часто;

− поскольку этот метод тестирования тесно связан с тестируемым приложением, инструменты для обслуживания каждого вида реализации / платформы могут быть недоступны.

Как следует из описания метод белого ящика резко контрастирует с методом черного ящика.

Для платформенных бизнес-приложений метод белого ящика следует применять на уровне модульного тестирования, причем применительно только к новому или измененному коду. Это позволит повысить эффективность тестирования.

Тестирования должно производиться разработчиком модуля.

2.1.2 Метод серого ящика

Тестирование по методу серого ящика - это метод тестирования ПО, который представляет собой комбинацию методов черного и белого ящиков.

Если в методе черного ящика внутренняя структура тестируемого элемента неизвестна тестировщику, в методе белого ящика полностью известна, то в методе серого ящика внутренняя структура ПО известна тестировщику частично.

Эти знания включают в себя доступ к внутренним структурам данных и алгоритмам для целей разработки тестовых случаев, хотя тестирование проводится на уровне пользователя или черного ящика.

Примером тестирования данного может служить анализ кодов для двух блоков / модулей (метод тестирования «белого ящика») для разработки тест-кейсов, а реальные тесты проводятся с использованием открытых интерфейсов (метод тестирования черного ящика).

Вполне объяснимо, что ему присущи достоинства и недостатки двух предыдущих методов.

Для платформенных бизнес-приложений метод серого ящика может применяться на уровне бета-тестировании, так как пользователь помимо проверки функционала бизнес-приложения, должен оценить возможности

программного инструментария платформы ИТ-решения для адаптации к потребностям конкретного предприятия.

2.2 Методологии тестирования на основе жизненного цикла бизнес-приложений

Тестирование является одной из ключевых стадий любого из вариантов жизненного цикла бизнес-приложения (ЖЦБП) [25].

Так, для успешной реализации бизнес-приложения необходимо выделить время и ресурсы для тестирования производительности и последующей оптимизации разработанного бизнес-приложения.

На рисунке 2.3 приведен пример модели типового ЖЦБП.

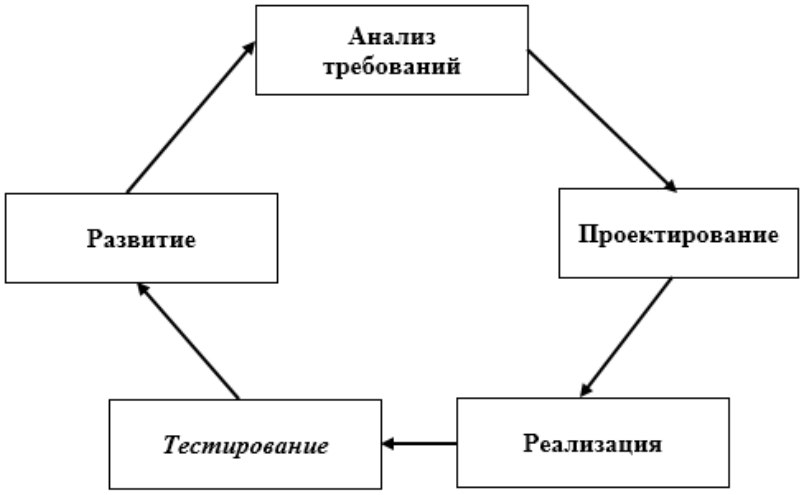

Рисунок 2.3 – Модель типового ЖЦБП

ЖЦБП состоит из нескольких четко обозначенных этапов работы, которые ИТ-специалисты используют для внедрения информационных систем.

Количество этапов, включенных в ЖЦБП, может составлять от 4 до 10.

В любом случае, цель разработки заключается в том, чтобы быстро спланировать, спроектировать, реализовать, протестировать и внедрить надежные системы.

Рассмотрим известные методологии тестирования, основанные на моделях ЖЦБП.

2.2.1 Каскадная модель жизненного цикла бизнес-приложения

Каскадная модель является одной из устаревших моделей, которую можно применять не только для разработки, но или для тестирования ПО [13].

Ее базовым принципом является последовательный порядок выполнения задач, что означает переход к следующему этапу разработки или тестирования только после того, как предыдущий был успешно завершен.

Эта модель подходит для небольших проектов и применима только в том случае, если все требования точно определены.

Главными достоинствами этой методологии являются экономическая эффективность, простота использования и управления документацией.

В этой модели процесс тестирования ПО начинается после завершения процесса разработки (рисунок 2.4).

На этой стадии все необходимые тесты переносятся с юнитов на системное тестирование для того, чтобы контролировать работу компонентов как по отдельности, так и в комплексе.

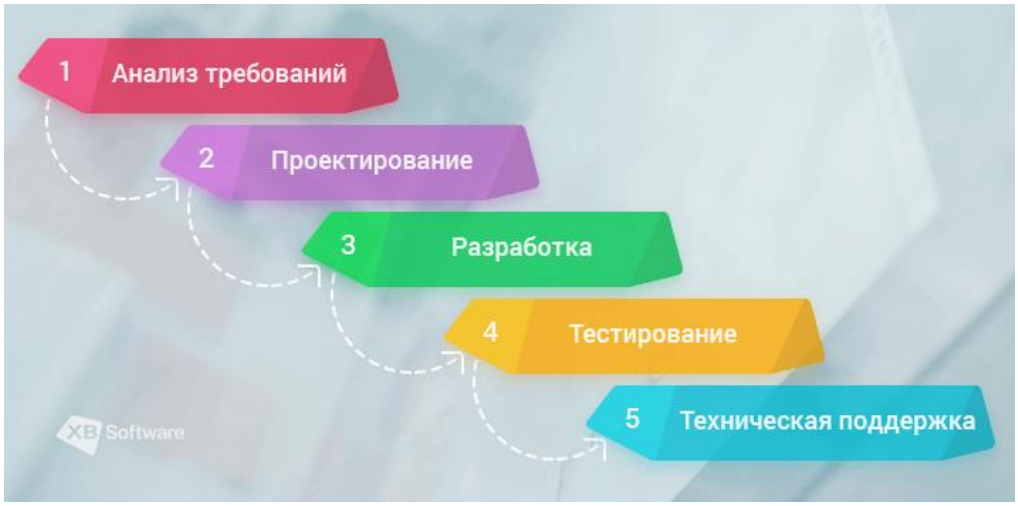

Рисунок 2.4 – Каскадная модель ЖЦБП

К недостаткам модели можно отнести высокую вероятность обнаружения критических ошибок в процессе тестирования. Это может потребовать полного изменения одного из компонентов системы и логики проекта, что невозможно осуществить в каскадной модели ЖЦБП.

2.2.2 V-образная модель жизненного цикла бизнес-приложения

Данная модель ЖЦБП, известная как модель валидации-верификации, также основана на прямой последовательности этапов.

Основным отличием с каскадной моделью ЖЦБП является планирование тестирования параллельно с соответствующей стадией разработки.

В этой модели процесс тестирования ПО начинается, как только определены требования и появляется возможность для статического тестирования ПО, что позволяет избежать возможных дефектов ПО на поздних стадиях.

Соответствующий план тестирования создается для каждого уровня разработки ПО, что определяет ожидаемые результаты, а также критерии входа и выхода для данного продукта.

Рисунок 2.5 представляет механизм разделения задач на две части. Задачи, относящиеся к дизайну и разработке, размещены слева. Соответственно задачи, относящиеся к тестированию ПО, размещены справа.

Рисунок 2.5 – Схема механизма разделения задач тестирования в V-образной модели ЖЦБП

Существенным недостатком представленной методологии тестирования является отсутствие готовых решений, позволяющих устранить дефекты ПО, обнаруженные на этапе тестирования.

2.2.3 Инкрементная модель жизненного цикла бизнес-приложения

Данная модель иначе называется мультикаскадной моделью тестирования ПО.

Рабочий процесс разделяется на некоторое количество циклов, каждый из которых также делится на модули. Каждая итерация добавляет определенный функционал к ПО.

Инкремент состоит из трех циклов: дизайн и разработка, тестирование и реализация.

В этой модели допускается одновременная разработка разных версий продукта.

Например, 1-я версия может проходить этап тестирования в то время, как 2-я еще находится на стадии разработки.

3-я версия в то же самое время может проходить этап дизайна. Этот процесс может продолжаться до самого завершения проекта (рисунок 2.6).

Рисунок 2.6 – Схема процесса тестирования для инкрементной модели ЖЦБП

Очевидно, что данная методология требует обнаружения максимально возможного количества ошибок в тестируемом ПО настолько быстро, насколько это возможно. Так же, как и фаза реализации, которая требует подтверждения готовности продукта к доставке к конечному пользователю.

Все эти факторы существенно увеличивают весомость требований к тестированию.

По сравнению с предыдущими методологиями, инкрементная модель имеет несколько важных преимуществ. Она более гибкая, изменение требований ведет к меньшим затратам, а процесс тестирования ПО является более эффективным, поскольку гораздо проще проводить тестирование и дебаггинг за счет использования небольших итераций. Тем не менее, стоит отметить, что общая стоимость все же выше, чем в случае каскадной модели.

2.2.4 Спиральная модель жизненного цикла бизнес-приложения

Спиральная модель представляет собой методологию тестирования ПО, основанную на инкрементном подходе и методе прототипирования.

Она состоит из четырех этапов: планирование, анализ рисков, разработка и оценка. Между этапами не должно быть заметного перерыва.

Тестирование ПО начинается еще на этапе планирования и длится до стадии оценки. Основным преимуществом спиральное модели является то, что первые результаты тестирования появляется незамедлительно после появления результатов тестов на третьем этапе каждого цикла, что помогает гарантировать корректную оценку качества. Тем не менее, важно помнить о том, что эта модель может быть довольно затратной и не подходит для маленьких проектов (рисунок 2.7)

Рисунок 2.7 – Спиральная модель ЖЦБП

Несмотря на то, что эта модель относится к устаревшим, она по-прежнему полезна как для тестирования, так и для разработки.

Следует отметить, что главная цель многих методологий тестирования ПО, включая спиральную модель, изменилась в последнее время.

Так, в настоящее время их применяют не только для поиска дефектов в приложениях, но и для выяснения вызвавших их причин.

Данный подход помогает разработчикам работать более эффективно и быстро устранять ошибки.

2.2.5 Гибкие методологии жизненного цикла бизнес-приложения

Сегодня господствует среди методологий проектирования бизнес-приложений господствуют разновидности методологии Agile.

Гибкое или Agile-проектирование ПО охватывает группу методологий ЖЦ ПО, основанных на итеративной разработке, когда требования и решения разрабатываются благодаря сотрудничеству между самоорганизующимися кросс-функциональными командами (рисунок 2.8).

Рисунок 2.8 – Модель Agile-методологии ЖЦБП

К Agile-методологиям относятся Scrum, XP, Kanban и др.

Одним из способов обеспечения производительности разработки программного обеспечения является объединение ролей в команде, в частности, сочетание разработки программного обеспечения с его тестированием.

Рассмотрим процесс тестирования в методологии Scrum.

В Scrum-команде нет ярко выделенных тестировщиков. Обычно тестирование проводится разработчиком с помощью модульного тестирования.

При этом Product Owner (владелец продукта) также часто участвует в процессе тестирования во время каждого спринта.

Некоторые проекты Scrum имеют специальные группы тестирования в зависимости от характера и сложности проекта.

Тестировщики выполняют следующие задачи:

1) На стадии планирования спринта тестировщик должен выбрать пользовательскую историю из журнала ожидания тестируемого продукта.

Тестировщик должен решить, сколько часов (усилий) потребуется, чтобы завершить тестирование для каждой из выбранных пользовательских историй.

Также он должен выяснить, каковы цели спринта, и внести свой вклад в процесс расстановки приоритетов проекта.

2) На стадии спринта тестировщик обеспечивает поддержку разработчиков в модульном тестировании.

Выполнение теста происходит в лаборатории, где и тестировщик, и разработчик работают рука об руку. Дефекты регистрируются в журнале управления дефектами, которые отслеживаются ежедневно.

Дефекты могут быть переданы и проанализированы во время встречи.

Дефекты повторно проверяются, как только они устранены и развернуты для тестирования.

Тестировщик посещает все ежедневные встречи и участвует в обсуждении.

В частности, он может предложить перенести невыполненное в текущем спринте задание в следующий спринт.

Тестировщик отвечает за разработку сценариев автоматизации.

Он планирует автоматизированное тестирование с помощью системы непрерывной интеграции, что особенно важно при коротких сроках разработки.

3) В процесс тестирования применяются механизмы автоматизированного тестирования.

Автоматическое тестирование - это процесс тестирования, выполняемый с помощью специализированного инструментария.

Преимущества автоматизированного тестирования бизнес-приложений:

− автоматизированное тестирование сокращает общее время внедрения и может также дать больше информации, поскольку данные тестирования автоматически сохраняются инструментами тестирования;

− автоматизированные тестовые сценарии позволяют быстро обнаружить любые ошибки или уязвимости и оперативно их исправить, прежде чем они станут большими проблемами;

− автоматизированное тестирование также может помочь гарантировать, что реализация будет идти в ногу с тем, чтобы новая ERP-система вашей организации была полностью запущена и работала в соответствии с бизнес-требованиями, изложенными в начале внедрения;

− бизнес-приложения обычно объединяют разрозненные источники данных в один централизованный пул данных. Автоматизированное тестирование позволит убедиться, что централизованная база данных остается активно подключенной к различным процессам, опосредованным бизнес-приложением и что его данные остаются защищенными;

− автоматизированное тестирование быстрое время отклика;

− автоматизированное тестирование снижает влияние человеческого фактора на результаты тестирования.

К недостаткам автоматизированного тестирования можно отнести затраты на внедрение ПО для его поддержки.

Таким образом, ключевым отличием Agile-методологий ЖЦБП является отсутствие жестких разграничений ролей членов команды разработчиков, а также применение механизмов автоматизированного тестирования.

На основании вышеизложенного можно сделать вывод об отсутствии единой методологии тестирования бизнес-приложений [14].

Выбор методологии тестирования зависит от типа проекта и используемой модели ЖЦБП, причем на разных этапах разработки можно использовать различные методологии.

Вместе с тем, представляется более целесообразным использовать для платформенных бизнес-приложений Agile-методологии тестирования, прежде всего благодаря применению автоматизированных механизмов тестирования.

Выводы к главе 2

1) При тестировании бизнес-приложения методом черного ящика необходимо обеспечить участие реальных пользователей и экспертов, знающих особенности реализуемых бизнес-функций, а также корректность входных данных.

2) Для повышения эффективности тестирования платформенных бизнес-приложений методом белого ящика, данный метод следует применять на уровне модульного тестирования, причем применительно только к новому или измененному коду.

3) Метод серого ящика рекомендуется применять на стадии бета-тестирования платформенного бизнес-приложения.

4) Следует констатировать отсутствии единой методологии тестирования бизнес-приложений. Вместе с тем, представляется более целесообразным использовать для платформенных бизнес-приложений Agile-методологии тестирования благодаря применению автоматизированных механизмов тестирования.

Глава 3 РАЗРАБОТКА МЕТОДИКИ ТЕСТИРОВАНИЯ ПЛАТФОРМЕННЫХ БИЗНЕС-ПРИЛОЖЕНИЙ

3.1 Постановка задачи на разработку методики тестирования

Анализ источников подтвердил отсутствие единого подхода к количественной оценке эффективности процесса тестирования бизнес-приложений.

Среди качественных критериев эффективности следует выделить исключение «человеческого фактора» из процесса тестирования бизнес-приложений. Это особенно важно, когда точность и корректность выполнения операций является критической для бизнеса [12].

Как было отмечено выше, одним из способов снижения влияния человеческого фактора является автоматизация тестирования. Вместе с тем, необходимо обеспечить полнофункциональное тестирование бизнес-приложений. Поэтому для постановки задачи на разработку методики тестирования платформенных бизнес-приложений, рассмотрим их функциональные и архитектурные особенности на примере решений на базе платформы 1С8.

На рисунках 3.1 и 3.2 изображены модель взаимодействия компонентов и «клиент-серверная» архитектура платформенного бизнес-приложения на базе 1С8, соответственно [4, 5].

Рисунок 3.1 – Модель взаимодействия компонентов платформенного бизнес-приложения на платформе 1С8: Предприятие

Рисунок 3.2 – Клиент-серверная архитектура бизнес-приложения на платформе 1С8: Предприятие

Одной из ключевых задач бизнес-приложений является обработка транзакций в режиме реального времени (OLTP), которая обуславливает повышенные требования к производительности серверов бизнес-приложения.

С учетом вышеизложенного сформулируем требования к методике тестирования платформенного бизнес-приложения:

1) обеспечение проверки бизнес-приложения с учетом его «клиент-серверной» архитектуры;

2) обеспечение проверки производительности серверов бизнес-приложения;

3) использование механизмов автоматизации конкретной технологической платформы для каждого вида тестирования.

Использование методики, построенной с учетом данных требований, позволит повысить эффективность процесса тестирования без снижения качества последнего.

3.2 Обзор и анализ существующих методик тестирования платформенных бизнес-приложений

3.2.1 Методика тестирования ERP-систем

Рассмотрим методику тестирования ERP и CRM-систем, в которой необходимо выполнить несколько типов тестирования, обеспечивающих проверку функциональности различных аспектов системы [16, 23].

Указанная методика состоит из следующих этапов:

1) Функциональное тестирование. Работая на основе точного перечня целей и определений, этот тип тестирования гарантирует, что каждая из функций в функциональной категории работает и отвечает потребностям организации.

2) Тестирование производительности - этот тип тестирования проверяет, насколько хорошо работает ERP-решение при взаимодействии с различными системами, с которыми оно взаимодействует (например, финансовые операции, обработка заказов на продажу, инвентаризация и т. д.).

Тестирование на проверку транзакций с высоким потоком данных, таких как транзакции, которые могут выполняться при самых высоких пиковых нагрузках. Цель тестирования - гарантировать, что ERP-система является достаточно устойчивой.

3) Интеграционное тестирование. Этот вид тестирования должен подтвердить, что в ERP-систему интегрированы все процессы, которые она должна выполнять. Выполняется тестирование отдельных модулей или компоненты ERP-системы, как группы.

В данном типе тестирования используются реальные сценарии, в которых реальные пользователи тестируют типичные ситуации, с которыми они сталкиваются в своей работе. Цель -убедиться, что все компоненты или модули в ERP-системе работают без сбоев.

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

Интеграционное тестирование может начаться на ранних этапах реализации и тестироваться по ходу работы.

Для повышения эффективности процесса тестирования рекомендуется использовать соответствующие средства автоматизации.

Недостатком данной методики является отсутствие в ней указаний по применению на каждом этапе конкретных методов и моделей, что ограничивает ее функциональные возможности.

3.2.2 Методика автоматизированного тестирования бизнес-приложений на основе система автоматизированного тестирования AQA

Предлагаемая методика предназначена для проверки качества и надежность новых версий системы «Галактика ERP» и других решений этой корпорации. Этот инструментарий тестирования также доступен партнерам «Галактики» и ИТ-службам предприятий-заказчиков [15].

По мнению разработчиков, тесты AQA дают возможность в автоматизированном режиме проверять производительность, масштабируемость и устойчивость работы системы под нагрузкой, проводить циклы тестирования функциональной преемственности (что чрезвычайно важно при выпуске новых версий, релизов, обновлений ПО), моделировать условия эксплуатации у заказчиков. Достигаются повышение качества поставляемой информационной системы «Галактика ERP», сокращение продолжительности разработки ее новых версий, повышение оперативности и качества услуг, оказываемых предприятиям-заказчикам.

Методика состоит из следующих направлений (этапов):

1) функциональное тестирование. Постоянно дополняемые библиотеки AQA-тестов используются при проверке каждой новой версии или обновления системы «Галактика ERP». Бизнес-процессы заказчика, переложенные на язык AQA-тестов, создают основу надежной эксплуатации системы на его предприятии;

2) параллельное тестирование системы «Галактика ERP» на каждой из СУБД, позволяющее сократить затраты на тестирование более чем наполовину;

3) нагрузочное тестирование и тестирование производительности;

4) стрессовое тестирование, позволяющие убедиться, что система будет работать надежно даже у самых крупных клиентов «Галактики».

Недостатком данной методики является ее специализация для решений, реализованных на платформе «Галактика», что снижает ее функциональные возможности.

3.2.3 Методика тестирования бизнес-приложений на платформе SAP

Структурная схема методики функционального тестирования бизнес-приложений, разработанных на популярной зарубежной платформе SAP представлена на рисунке 3.3 [22].

Для автоматизации тестирования используется программное решение IBM Rational Functional Tester.

Предлагаемая методика не критична к уровню подготовки тестировщика.

Так, непрофессиональный тестировщик может записывать сценарии функционального тестирования, которые генерируются как упрощенные сценарии тестирования.

Тестировщику не требуются знания программирования для редактирования сценариев функционального тестирования. Он может использовать визуальные элементы приложения для вставки контрольных точек, команд, управляемых данными, и дополнительных элементов управления для тестирования.

Профессиональный тестировщик со знанием программирования на Java или Visual Basic может записывать сценарии функционального тестирования, или создавать сценарии тестирования вручную. Он может использовать карты тестовых объектов для обновления объектов и вставки дополнительных объектов для тестирования.

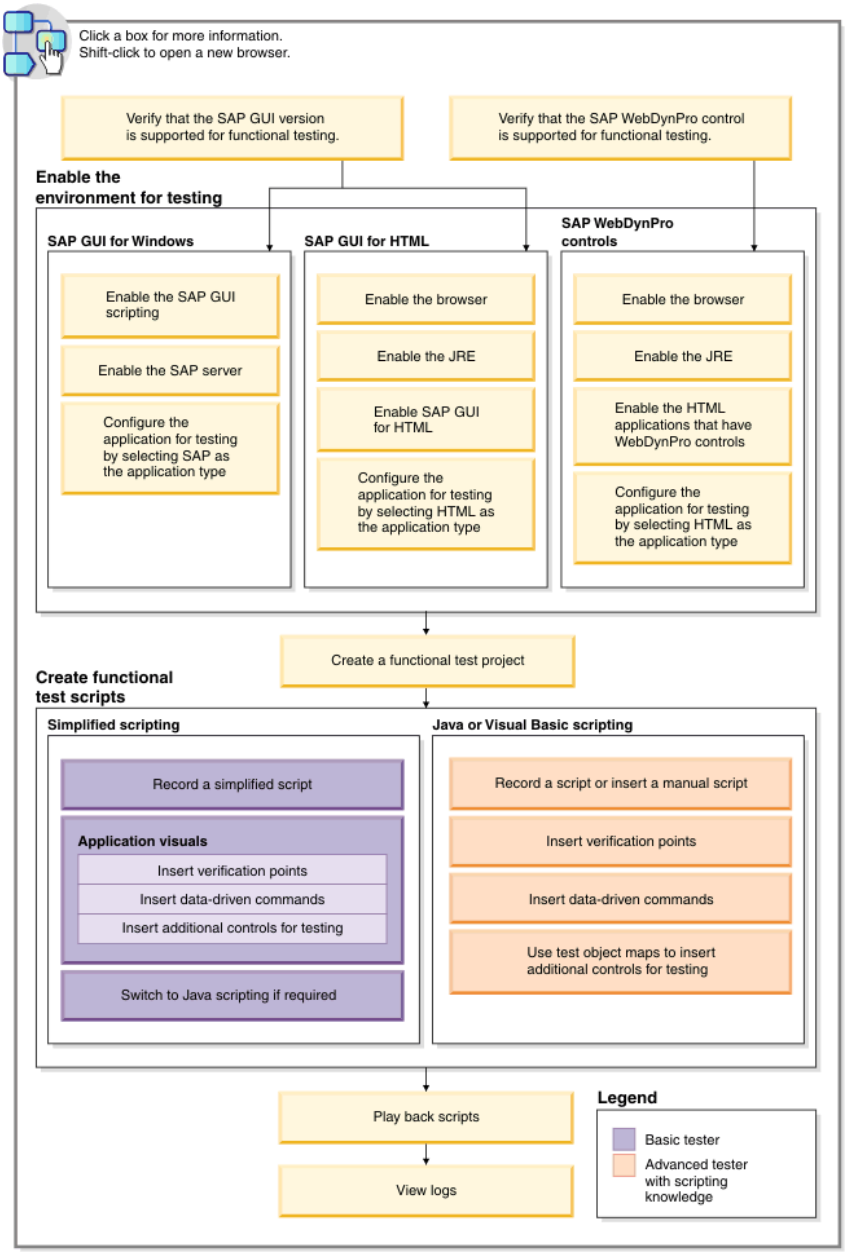

Рисунок 3.3 – Структурная схема методики тестирования бизнес-приложений на платформе SAP

3.2.4 Методика интеграционного тестирования бизнес-приложений 1С

Методика ориентирована на повышение эффективности интеграционного тестирования бизнес-приложений 1С, построенных на основе технологии СОА.

Для решения данной задачи используется следующий подход: для веб-сервиса, который может быть подвержен изменениям создается набор т.н. «дымовых» (smoke) тестов.

Дымовые тесты проверяют, что основные методы сервиса не изменились и обращение к ним не вызывает ошибок.

Как было отмечено выше, в интеграционном тестировании используется механизм программных заглушек («мок», mock).

Мок-сервис позволяет переопределить реальный сервис и подставить вместо него упрощенную реализацию, которая работает нужным разработчику образом и дает доступ к собственным данным и настройкам.

Для создания мок-сервисов предлагается использовать программу SoapUI.

Программа SoapUI позволяет на основании WSDL-схемы веб-сервиса создать мок-сервис и опубликовать его на собственном веб-сервере, который умеет запускать.

Тестируемое приложение 1С больше не будет обращаться к удаленному сервису, а получит ответ, который разработчик заранее подготовил и опубликовал из SoapUI (рисунок 3.4).

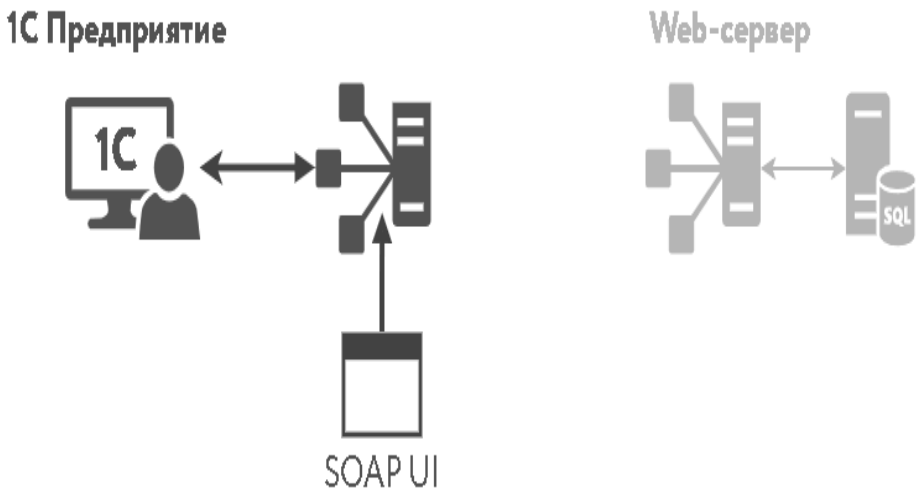

Рисунок 3.4 – Схема процесса тестирования

Следует отметить, что данная методика имеет некоторые функциональные ограничения, так как предназначена для процесса тестирования бизнес-приложений 1С, интегрированных с различными веб-сервисами по технологии СОА.

3.2.5 Анализ известных методик тестирования платформенных бизнес-приложений

Анализ известных методик тестирования платформенных бизнес-приложений позволил сделать следующие выводы:

1) Основной особенностью исследованных методик является привязка используемых в них видов и методов тестирования к программной архитектуре технологической платформы, на базе которой реализовано тестируемое бизнес-приложений.

Иными словами, методики тестирования платформенных бизнес-приложений не являются универсальными.

2) Процесс тестирования не должен быть затратным и трудоемким, к квалификации тестировщику не должны предъявляться особые требования по знанию встроенного платформенного языка, а его участие в процессе тестирования должно быть по возможности ограничено. Последнее достигается за счет применения специализированных средств автоматизации тестирования.

3) Основным видом тестирования, применяемым в указанных методиках, является функциональное тестирование, которое, как правило, выполняется на модульном и системном уровнях. При этом в доступных материалах по рассмотренным методикам нет четких описаний методов и моделей тестирования, используемых в них.

Таким образом, для обеспечения повышения эффективности тестирования платформенного бизнес-приложения необходимо использовать методику, разработанную с учетом функциональных и архитектурных особенностей технологической платформы, на базе которой реализовано данное приложение, и ориентирована на применение соответствующих средств автоматизации.

3.3 Методика тестирования бизнес-приложений 1С8

На основе анализа известных подходов к тестированию бизнес-приложений, с учетом их функциональных и архитектурных особенностей

разработана методика тестирования бизнес-приложений, реализованных на базе технологической платформы «1С: Предприятие 8».

В предлагаемой методике тестирование бизнес-приложения 1С8 выполняется в следующим порядке:

1) Тестирование конфигурации бизнес-приложения.

Включает в себя модульный и системный уровни тестирования бизнес-приложения.

2) Тестирование серверной части бизнес-приложения.

Состоит из функционального тестирования системы управления базами данных (СУБД), используемой в качестве сервера баз данных (СБД), и нагрузочное тестирование всех серверов, используемых в бизнес-приложении.

UML диаграмма вариантов использования предлагаемой методики тестирования представлена на рисунке 3.5.

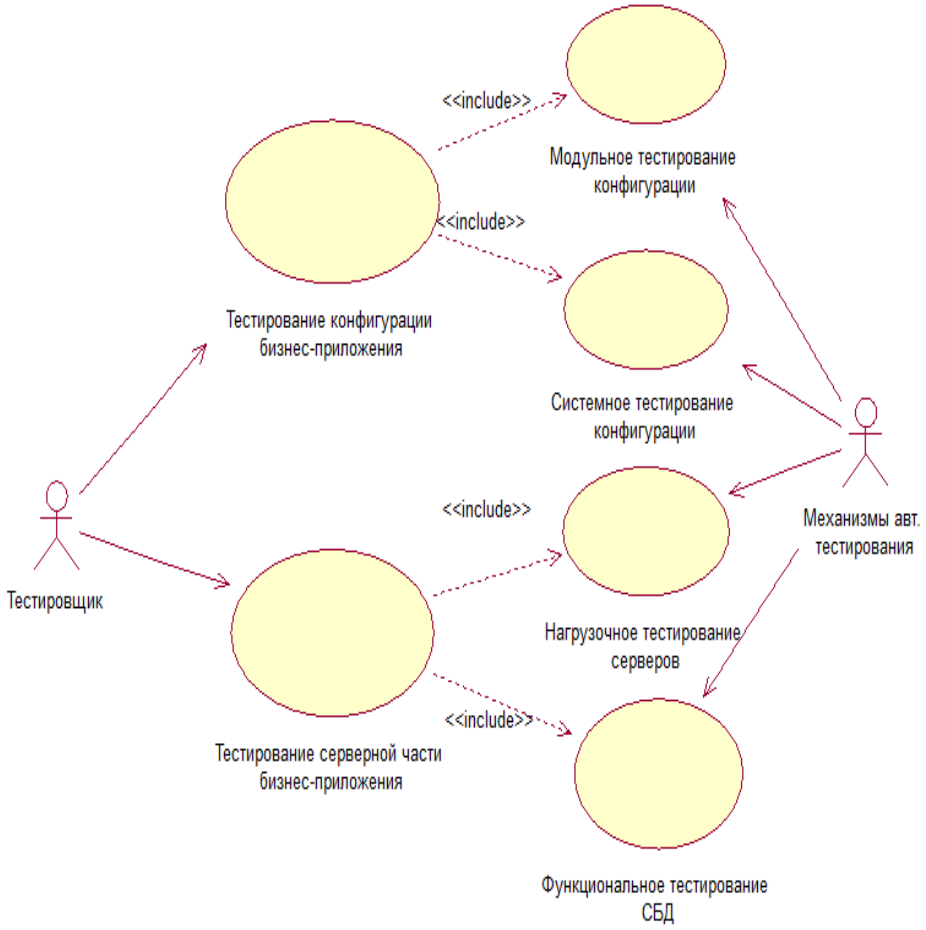

Рисунок 3.5 – Диаграмма вариантов использования методики тестирования бизнес-приложений 1С8

Рассмотрим основные положения предлагаемой методики.

3.3.1 Тестирование конфигурации бизнес-приложения 1С8

Тестирование конфигурации - это тип тестирования ПО, который проверяет бизнес-приложение с несколькими комбинациями программного и аппаратного обеспечения, чтобы определить оптимальные конфигурации, с которыми система может работать без каких-либо недостатков или ошибок.

Тестирование конфигурации включает проверку поведения бизнес-приложения в различных средах.

Этот тип тестирования определяет непосредственное или долгосрочное влияние изменений конфигурации на поведение и производительность системы.

Как правило, число возможных конфигураций на стадии проектирования бизнес-приложения может быть слишком велико для тестирования. Уже один этот факт означает, что на этапе планирования усилий по тестированию конфигурации необходимо определить только те конфигурации, которые будут поддерживаться.

Приоритеты должны быть установлены на основе ожидаемой пользовательской базы и рисков, связанных со скрытыми ошибками в определенных конфигурациях.

Как следует из вышеизложенного, это длительный процесс тестирования, так как он должен проводиться после каждого обновления конфигурации бизнес-приложения и требует много времени для установки и удаления различного программного обеспечения, которое будет использоваться для тестирования.

В зарубежных источниках используется понятие базовой матрицы конфигураций (рисунок 3.6).

Тестирование конфигурации позволяет выявить ошибки, которые не являются критичными для функционирования прикладного решения в принципе, но наличие которых может существенно снизить скорость работы

прикладного решения или даже привести к возникновению ошибок при работе в некоторых специальных режимах.

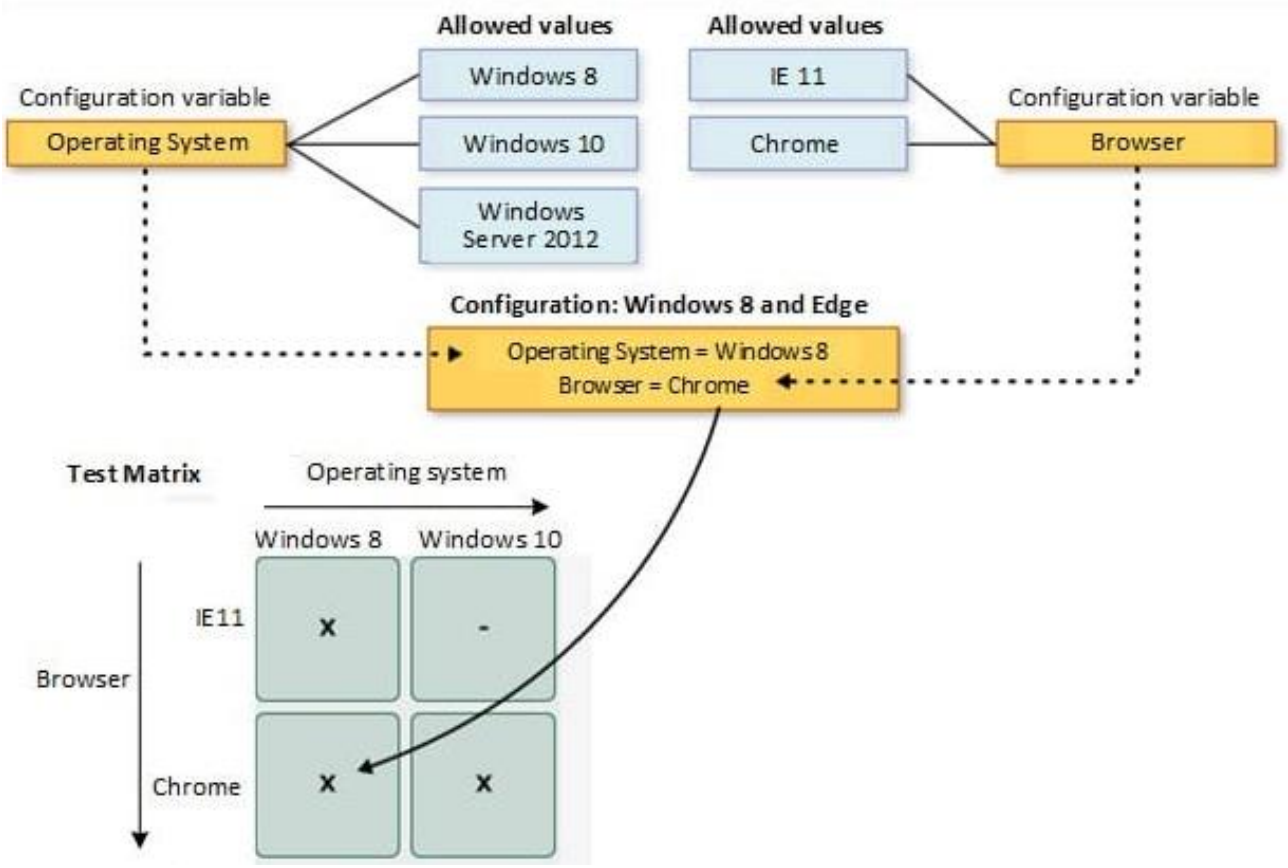

Рисунок 3.6 – Пример матрицы тестируемых конфигураций

В 1с8 выполнение данных проверок осуществляется, например, для проверки конфигурации перед поставкой заказчику, перед выпуском тиражного решения, для проверки после массированного удаления объектов или после объединения конфигураций [10].

Функциональное тестирование конфигурации бизнес-приложения 1С8 включает в себя модульное и системное тестирование.

Модульное тестирование является достаточно простым видом тестирования применительно к бизнес-приложениям 1С8, так как предполагает проведение тестирования только тех компонентов конфигурации, которые были созданы или модифицированы разработчиками бизнес-приложения.

Функциональное тестирование конфигурации 1С8 выполняется по методу сценарного тестирования.

Как правило, тестирование конфигурации бизнес-приложения 1С8 включает:

- проверку логической целостности конфигурации;
- поиск некорректных ссылок;
- синтаксический контроль модулей;
- логическую проверку модулей и др.

Для функционального тестирования конфигурации бизнес-приложения необходимо использовать механизм автоматизированного тестирования технологической платформы «1С: Предприятие 8».

3.3.2 Тестирование серверной части бизнес-приложения 1С8

Тестирование серверной части бизнес-приложения включает в себя следующие уровни:

1) Функциональное тестирование сервера баз данных (СБД).

Сервер баз данных – это серверная СУБД, используемая бизнес-приложением.

Для функционального тестирования сервера баз данных предлагается использовать модель тестирования реляционной СУБД, предложенную С. Амблером [24].

Модель построена с точки зрения единой базы данных ИС.

Пунктирные линии обозначают границы угроз, которые необходимо учитывать, как внутри базы данных (тестирование по методу белого ящика), так и на уровне интерфейса с базой данных (тестирование по методу черного ящика) (рисунок 3.7).

По мнению экспертов Agile-метологий существует несколько причин, из-за которых необходимо разработать комплексную стратегию тестирования СУБД:

- данные являются важным корпоративным ресурсом;
- В СУБД реализована критически важная бизнес-функциональность;

− отсутствие реальных механизмов проверки качества данных на стадии разработки бизнес-приложения;

− многие методы эволюционной разработки, в частности рефакторинг базы данных (БД), основаны на идее, что должна быть возможность определить, было ли что-то в БД повреждено при внесении изменений.

Рисунок 3.7- Модель тестирования СУБД

В таблице 3.1 представлены объекты функционального тестирования СУБД.

Для функционального тестирования серверной СУБД используется метод «Разработка через тестирование» (Test-Driven Development, TDD).

TDD - это эволюционный подход к разработке, который сочетает метод разработки в первую очередь (Test-First Development, TFD) с тестированием и рефакторингом.

В рассматриваемом контексте, рефакторинг – это процесс изменения работающего кода бизнес-приложения.

Таблица 3.1 - Объекты функционального тестирования СУБД

Тестирование интерфейса (метод черного ящика)	Внутреннее тестирование БД (метод белого ящика)
• Отображение мета-данных; • Значения входных данных; • Значения выходных данных запросов, хранимых процедур, представлений и т.д.	• Скаффолдинг (метод метапрограммирования для взаимодействия с БД, например, триггеры или обновляемые представления); • Типовые модульные тесты для хранимых процедур, функций и триггеров; • Существующие тесты для элементов схемы БД (таблицы, процедуры и т.д); • Определения представлений; • Правила ссылочной целостности; • Значения по умолчанию для столбца; • Инварианты данных для одного столбца; • Инварианты данных, включающие несколько столбцов

Алгоритм данного метода в виде диаграммы деятельности UML представлен на рисунке 3.8.

Для повышения эффективности тестирования следует использовать механизм автоматизированного тестирования.

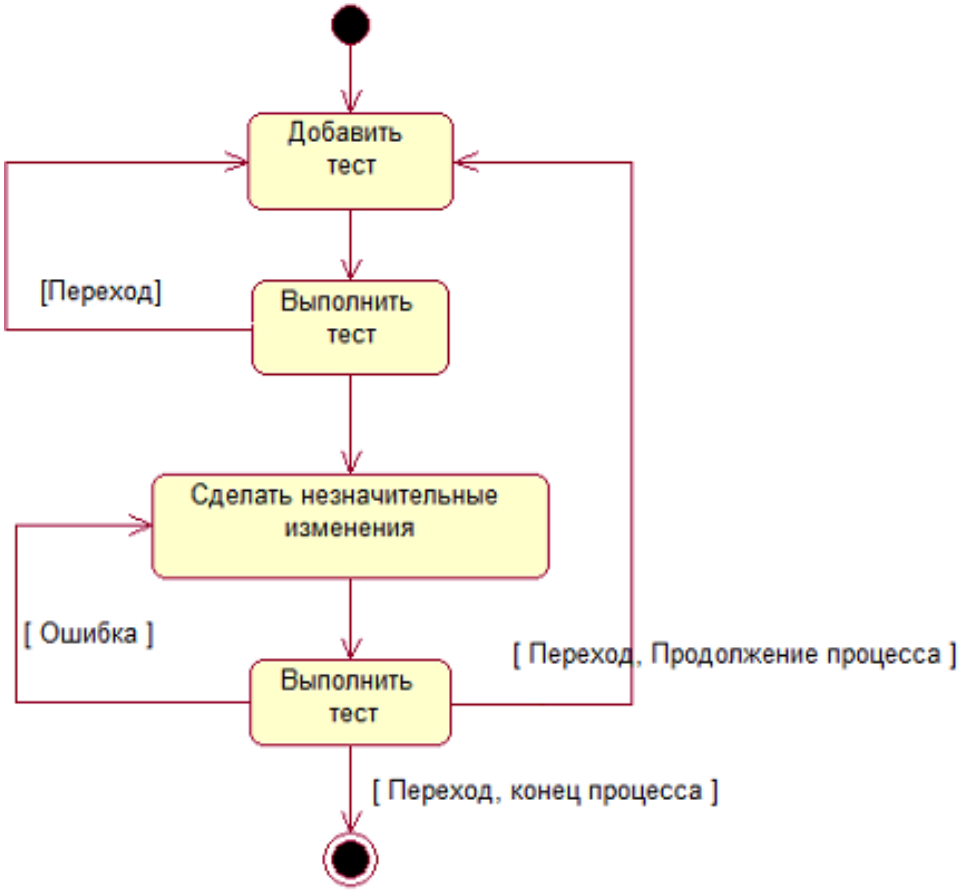

Рисунок 3.8 – Алгоритм метода TDD

2) Нагрузочное тестирование серверов бизнес-приложения.

Обязательной операцией при любом внедрении или изменении существующей информационной системы (ИС) является оценка необходимого быстродействия системы и планирование необходимых вычислительных ресурсов для ее реализации.

Существует достаточное количество способов оценить необходимую для достижения требуемой производительности конфигурацию программного и аппаратного обеспечения. Все эти способы могут применяться в процессе выбора, но потребитель должен понимать их области применения и ограничения.

Для решения данной задачи используется, например, нагрузочное тестирование.

Нагрузочное тестирование серверов является одним из обязательных видов тестирования платформенного бизнес-приложения.

Это обусловлено тем, что промышленные технологические платформы, как правило, могут работать с несколькими СУБД.

Например, платформа «1С: Предприятие 8.3» поддерживает СУБД Microsoft SQL Server, PostgreSQL, IBM DB2 и Oracle Database.

При выборе конкретной СУБД или замене одной СУБД на другую из установленного перечня результаты нагрузочного теста являются одними из ключевых для принятия решения по данной проблеме.

Другим очень важным объектом тестирования является сервер приложений «1С: Предприятия 8» или иначе «Сервер 1С».

Следует также учесть, что современные бизнес-приложения 1С8 работают в режиме терминального доступа.

В этом случае нагрузочное тестирование поможет разработчикам оценить производительность внедренных терминальных ИТ-решений.

Процесс проведения нагрузочного тестирования любой ИС обычно включает следующие этапы:

- сбор требований;
- анализ текущей ситуации, если эталонная ИС уже существует;
- подготовка тестовой системы;
- расчет требований к оборудованию;
- подготовка тестовой среды;
- обеспечение технологического качества ИС на тестовой площадке;
- формирование технологических требований и изменений по результатам проведенного нагрузочного тестирования;
- переход к этапу внедрения ИС.

Все последующие этапы обязательно выполняются с учетом результатов предыдущих этапов.

Указанные этапы не могут быть проведены параллельно либо в другом порядке.

Параллельность выполнения работ может быть достигнута организационными мерами в рамках каждого из этапов.

На практике нагрузочное тестирование серверов для платформенных решений организуется на основе методов автоматизированного тестирования, разработанных специально для данной платформы.

Это способствует значительному повышению эффективности указанного процесса.

Для нагрузочного тестирования серверов в 1С8 предлагается использовать автоматизированный нагрузочный тест ТРС-1С, известный также как тест Гилева [11].

В основу данного теста положен подход к оценке производительности сервера, основанный на том, что последняя определяется не загруженностью и очередями к процессору, а способностью выполнить количество операций в единицу времени.

В тесте используются два основных типа тестирования: компонентное и интегральное.

При компонентном тестировании проводится тестирование отдельных компонентов решения, начиная от производительности процессоров или подсистем хранения информации до тестирования производительности сервера в целом, но без полезной нагрузки в виде того или иного бизнес-приложения.

Интегральный подход характеризуется оценкой производительности решения в целом, включая программную и аппаратную части.

При этом может использоваться как бизнес-приложение, которое будет использовано в конечном решении, так и некоторые модельные приложения, эмулирующие некоторые стандартные бизнес-процессы и нагрузки.

Другими словами, и настройки СУБД, и настройки ОС, и оборудование оказывают влияние на общий командный результат.

Тест оценивает количество работы в единицу времени в одном потоке и подходит для оценки скорости работы однопоточных нагрузок, включая скорость графического интерфейса, перепроведения документов и т.п. Тест

относится к разделу универсальных интегральных кроссплатформенных тестов. Даже более того, он применим для файлового и клиент-серверного вариантов реализации бизнес-приложения.

Тест работает для всех СУБД, поддерживаемых 1С. Универсальность позволяет делать обобщенную оценку производительности, не привязываясь к конкретной типовой конфигурации платформы.

Узким местом теста является то, что он не делает диагностики. Тест не оценивает возможности оборудования для коллективной работы в условиях интенсивных блокировок.

Выводы к главе 3

1) Для повышения эффективности тестирования платформенного бизнес-приложения должна использоваться методика, при разработке которой помимо специфики технологической платформы приняты во внимание реализация бизнес-приложения в архитектуре «клиент-сервер» и повышенные требования к производительности используемых серверов.

2) Предлагаемая методика тестирования бизнес-приложения 1С8 включает в себя тестирование конфигурации бизнес-приложения и его серверной части.

3) Представленная методика основана на применении механизмов автоматизированного тестирования системы «1С: Предприятие 8».

4) Для функционального тестирования СУБД, используемой в качестве СБД, рекомендуется использовать регрессионное тестирование по методу TDD.

5) Для нагрузочного тестирования серверов в 1С8 рекомендуется использовать автоматизированный нагрузочный тест ТРС-1С.

Глава 4 АПРОБАЦИЯ МЕТОДИКИ ТЕСТИРОВАНИЯ ПЛАТФОРМЕННЫХ БИЗНЕС-ПРИЛОЖЕНИЙ

Апробация разработанной методики проведена на технологической платформе «1С: Предприятие 8.3», работающей с СУБД PostgreSQL.

4.1 Апробация методики тестирования конфигурации бизнес-приложения 1С8

Для проведения эксперимента, в типовую конфигурацию добавлен новый справочник.

Алгоритм тестирования конфигурации бизнес-приложения 1С8 представлена на рисунке 4.1.

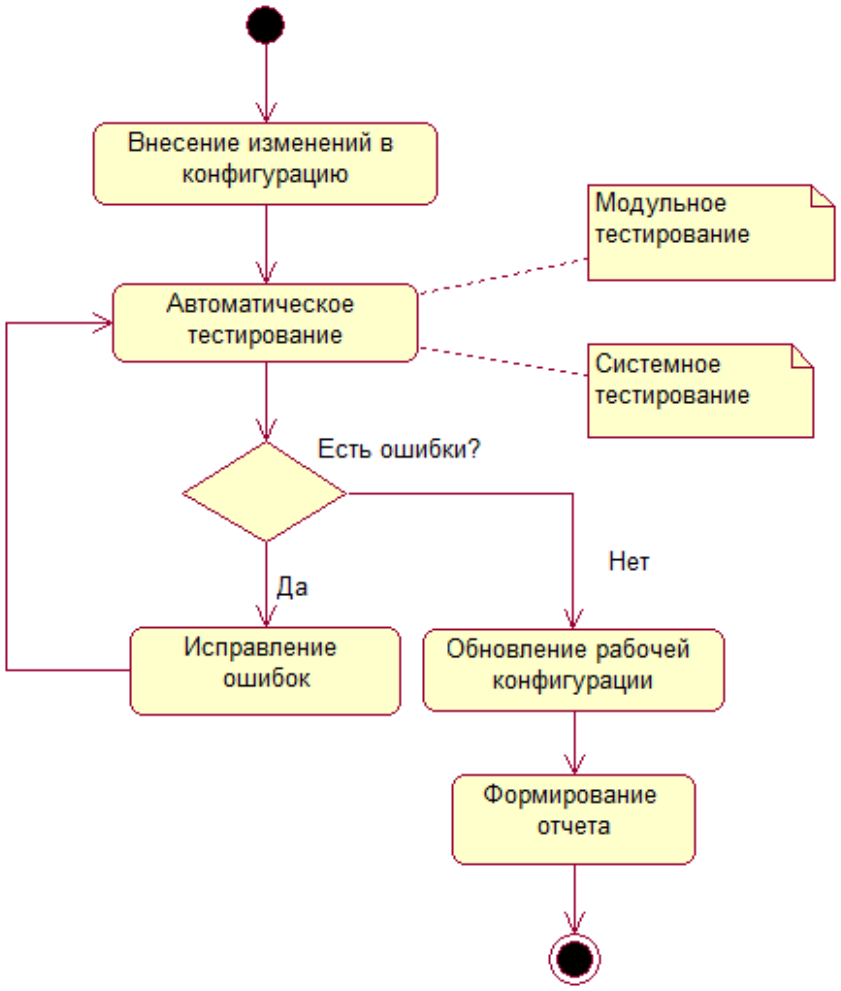

Рисунок 4.1- Алгоритм тестирования конфигурации бизнес-приложения 1С8

Для автоматизации процесса тестирования конфигурации бизнес-приложения используется встроенный механизм автоматизации платформы «1С: Предприятие 8.3», модель которого изображена на рисунке 4.2.

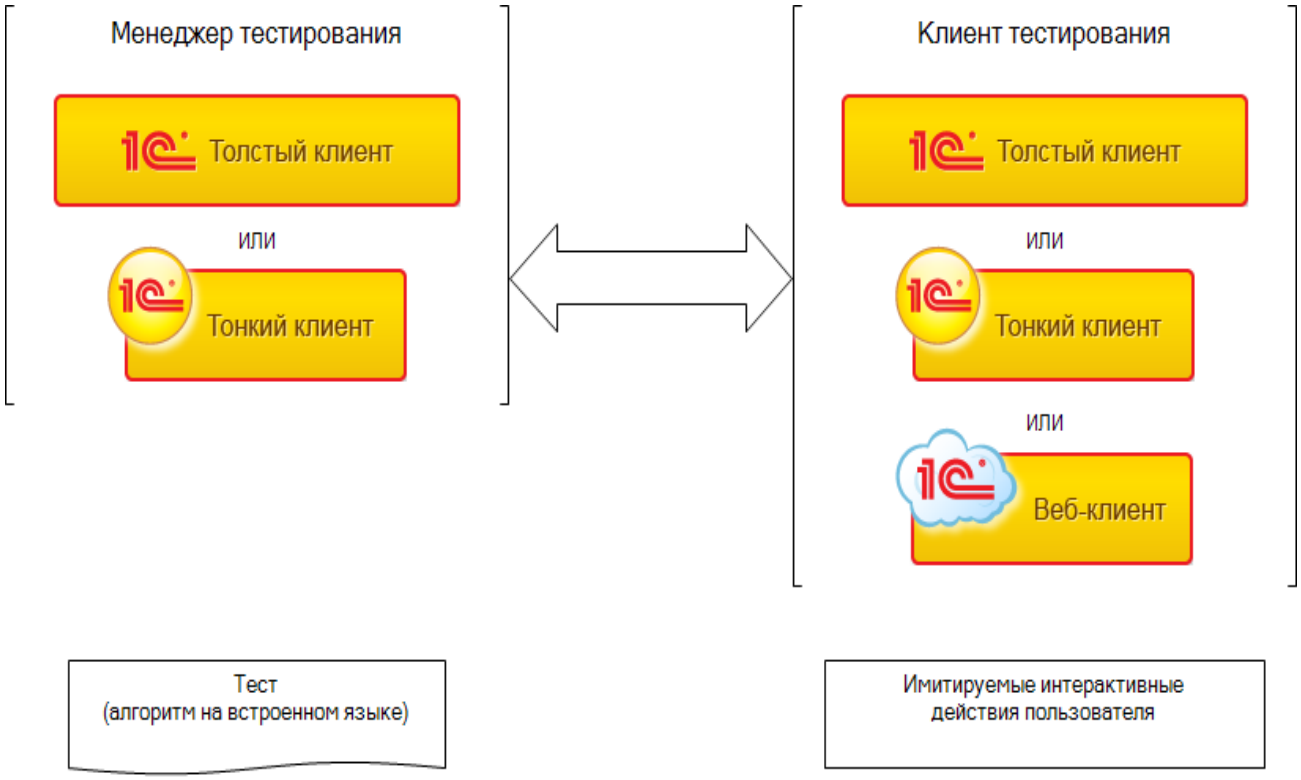

Рисунок 4.2 - Модель механизма автоматизированного тестирования платформы «1С: Предприятие 8.3»

Для автоматизации процесса тестировании используются два вида клиентских приложений – менеджер тестирования и клиент тестирования. Менеджер тестирования устанавливает связь с клиентом тестирования и выполняет сценарий тестирования.

Сценарий тестирования – это код на встроенном языке, в котором описывается последовательность выполняемых интерактивных действий. Для этого во встроенный язык добавлены новые объекты, которые на абстрактном уровне описывают интерфейс приложения (оперируя понятиями окна, формы, элементов управления и т.п.), а также описывают действия пользователей (навигация по конфигурации, ввод данных и т.п.).

Менеджер тестирования может быть толстым или тонким клиентом. Клиент тестирования – толстым, тонким клиентом или веб-клиентом.

Менеджер тестирования может быть подключен к нескольким клиентам тестирования, а клиент тестирования может быть подключен только к одному менеджеру.

Автоматизированное тестирования выполняется по алгоритму, который состоит из следующих шагов:

Шаг 1. Создается сценарий тестирования - разрабатывается внешняя или встроенную в конфигурацию обработка, в которой будут последовательно описаны выполняемые этапы тестирования.

Шаг 2. Запускается менеджер тестирования.

Шаг 3. Запускается клиент (клиенты) тестирования.

Шаг 4. В менеджере тестирования запускается на исполнение созданная обработка. Необходимо убедиться в выполнении запрограммированных действий на клиенте.

Тестируемое приложение описывается набором объектов встроенного языка, которые используются для написания сценария:

- ТестируемоеПриложение;
- ТестируемоеОкноКлиентскогоПриложения;
- ТестируемыйКомандныйИнтерфейсОкна;
- ТестируемаяГруппаКомандногоИнтерфейса;
- ТестируемаяКнопкаКомандногоИнтерфейса;
- ТестируемаяФорма;
- ТестируемоеПолеФормы;
- ТестируемаяГруппаФормы;
- ТестируемаяКнопкаФормы;
- ТестируемаяТаблицаФормы;
- ТестируемаяДекорацияФормы.

Примеры сценария теста, выполняющего проверку версии конфигурации в соответствии с переданным списком требований, представлены на рисунке 4.3.

```
#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 4.3 – Сценарий проверки конфигурации бизнес-приложений

```
        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");
```

Продолжение рис. 4.3

```
        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("Заг
оловок"));

        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("Ош
ибка"));
                КонецЦикла;
            ТабДокумент.Показать();
            Возврат ТабДокумент;
        КонецФункции

#КонецОбласти

#КонецЕсли
```

Окончание рис. 4.3

Сценарии тестирования реализованы в виде внешних обработок, хранящихся в файле с расширением .EPF (рисунок 4.4).

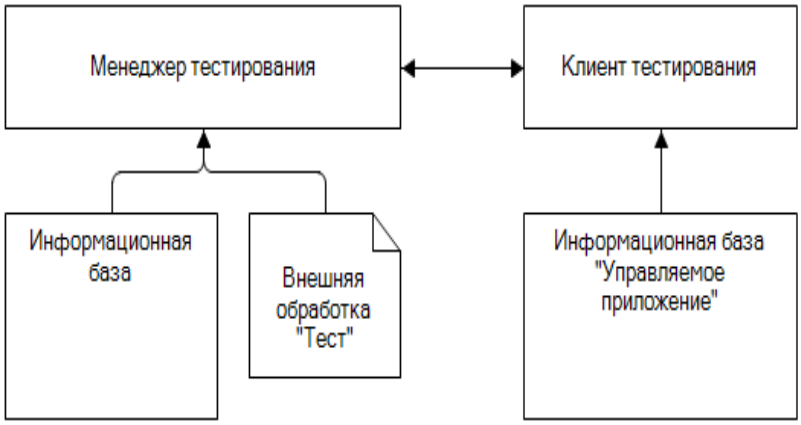

Рисунок 4.4 – Модель автоматизированного тестирования с помощью внешней обработки

Результаты тестирования модульного тестирования справочника бизнес-приложения (Приложение А), представлены в отчете ниже (таблица 4.1).

Таблица 4.1 - Отчет модульного тестирования справочника

Номер этапа	Описание	Результат тестирования
1.	проверка конфигурации на соответствие утвержденному проекту разработки и внедрения бизнес-приложения	соответствует
2.	модульное тестирование конфигурации бизнес-приложения	ошибки устранены
3.	системное тестирование конфигурации бизнес-приложения	ошибки не обнаружены
4.	проверка качества информационного обмена между отдельными модулями бизнес-приложения	соответствует требованиям

Разработчики бизнес-приложений, как правило, создают библиотеки эффективных тестовых сценариев и используют их в своей дальнейшей работе.

4.2 Апробация методики тестирования серверной части бизнес-приложения 1С8

Тестирование информационной базы (ИБ) бизнес-приложения производится по алгоритму, представленному на рисунке 4.5.

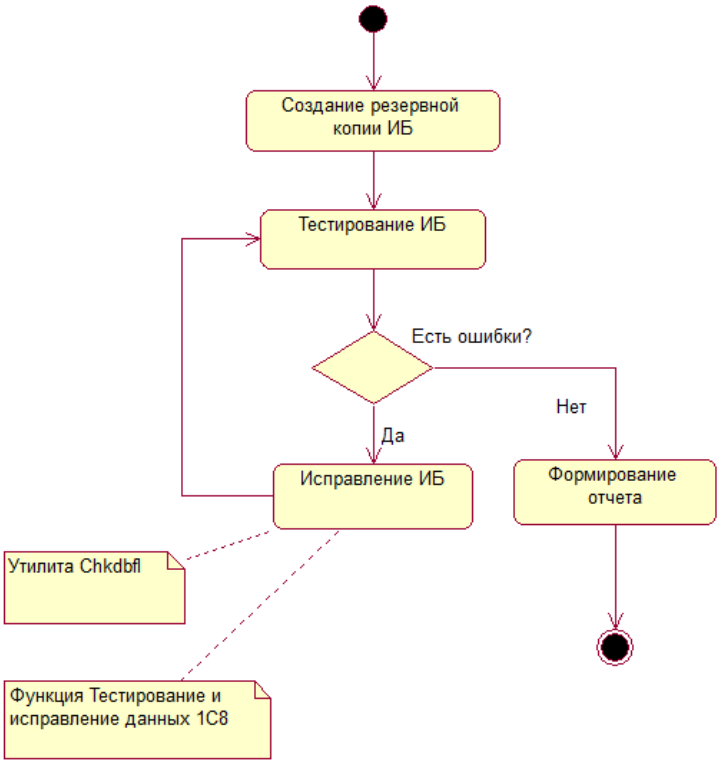

Рисунок 4.5 – Алгоритм тестирования информационной базы бизнес-приложения 1С8

Для автоматизации данного процесса используется встроенная в конфигуратор функция «Тестирование и исправление» (рисунок 4.6).

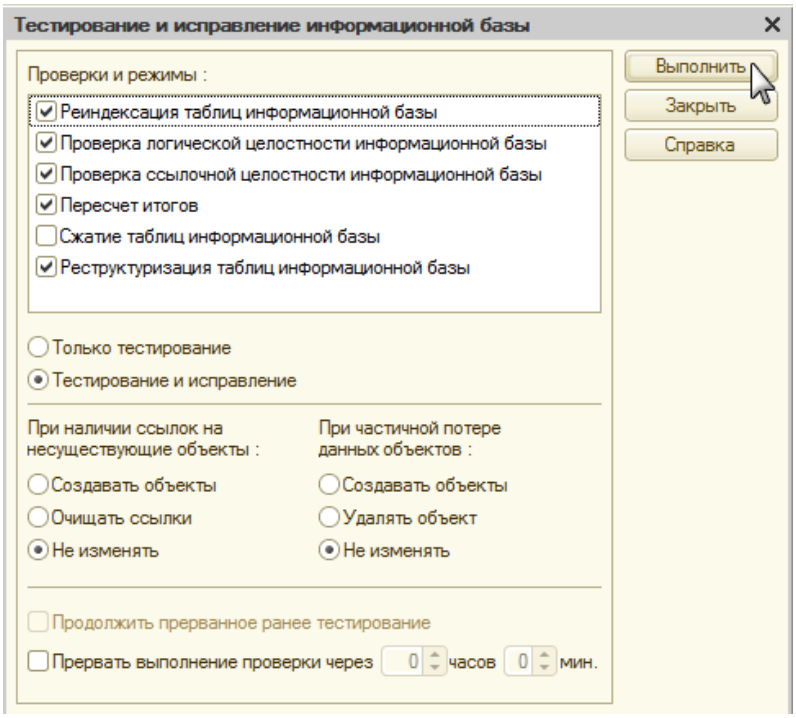

Рисунок 4.6 – Окно функции «Тестирование и исправление»

Данная функция реализует следующие способы проверки ИБ:

– проверка логической целостности —комплекс проверок, целью которых является проверка логики базы данных;

– реиндексация таблиц — оптимизация таблиц за счёт перестроения их индексов для повышения быстродействия базы данных;

– проверка ссылочной целостности — комплекс проверок, в ходе которых отыскиваются несуществующие ссылки на объекты;

– пересчёт итогов — проверка, в ходе которой происходит пересчёт итогов таблиц в регистрах накопления;

– реструктуризация таблиц ИБ;

– сжатие таблиц базы (для файловой ИБ).

Для исправления файловых ошибок структуры файловой ИБ применяется утилита chkdbfl.

С помощью теста TPC-1C выполнено нагрузочное тестирование терминального сервера, сервера приложений 1С (сервер 1с) и СУБД PostgreSQL. Для автоматизации тестирования использован инструментарий, предоставляемый на сайте разработчика теста.

На рисунках 4.7-4.9 представлены типичные нагрузки серверов бизнес-приложения.

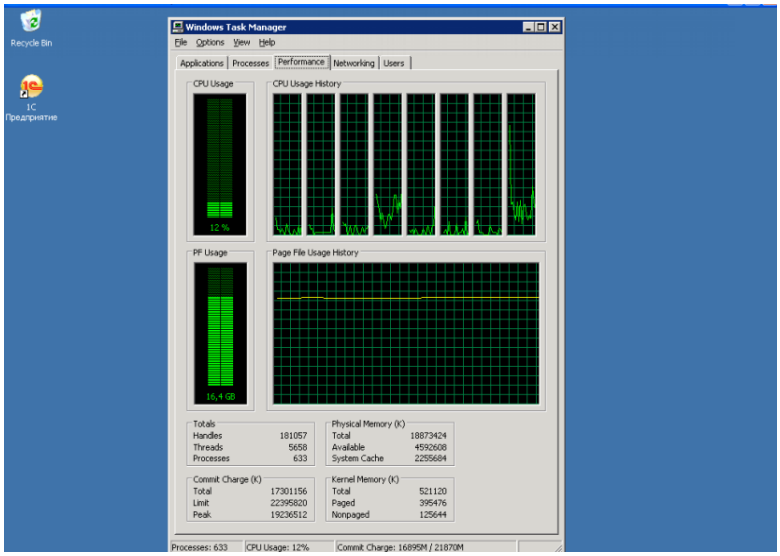

Рисунок 4.7 – Типичная нагрузка терминального сервера (100 пользователей)

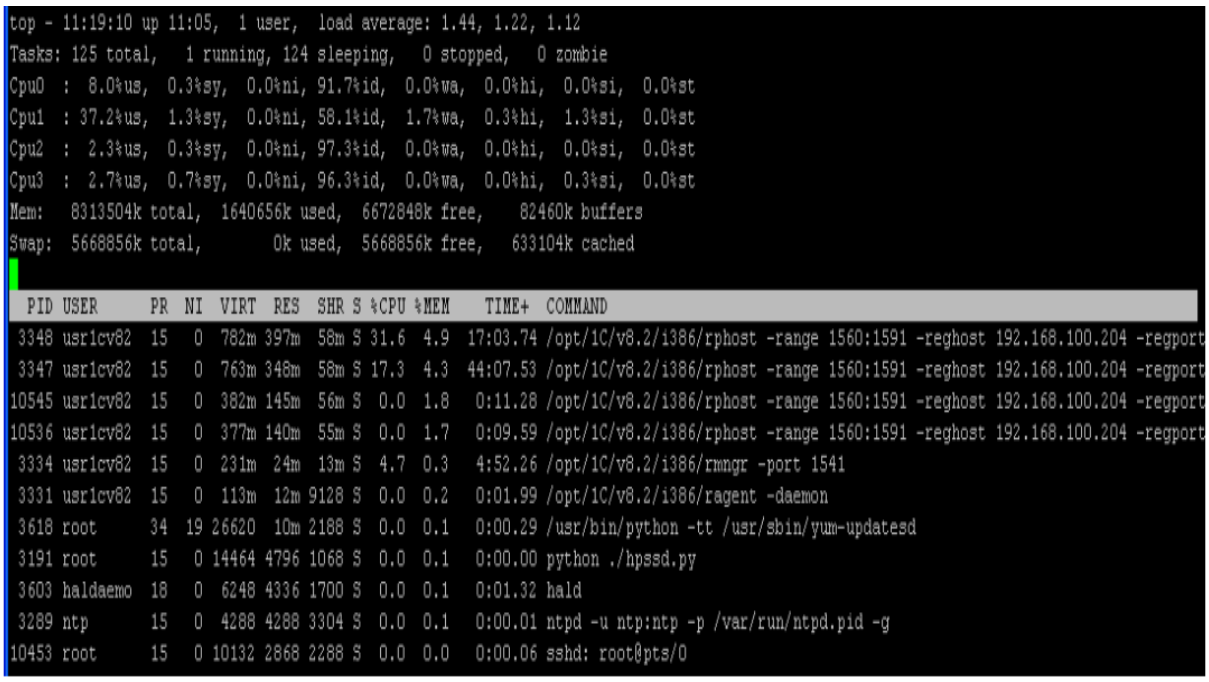

Рисунок 4.8 – Типичная нагрузка сервера 1С

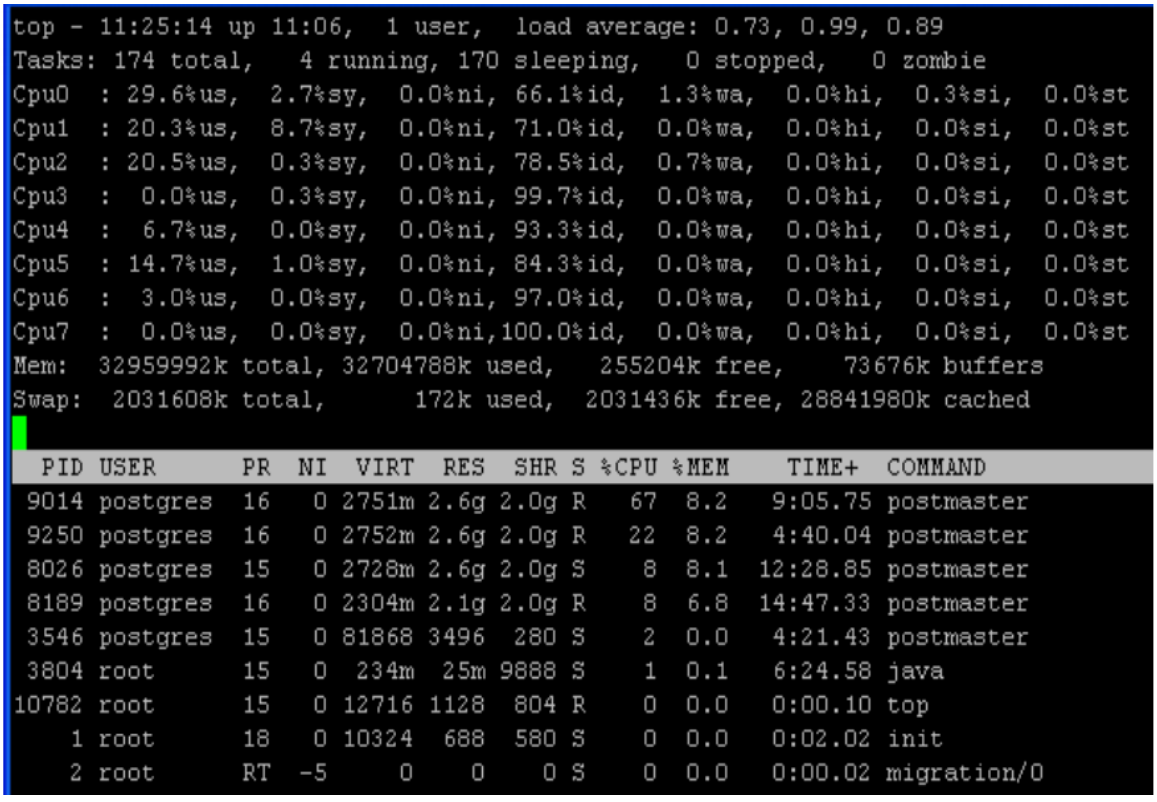

Рисунок 4.9 – Типичная нагрузка на СУБД PostgreSQL

Результаты теста TPC-1C выводятся на экран в виде диаграммы, цвет которой определяет оценку производительности серверов (рисунок 4.10).

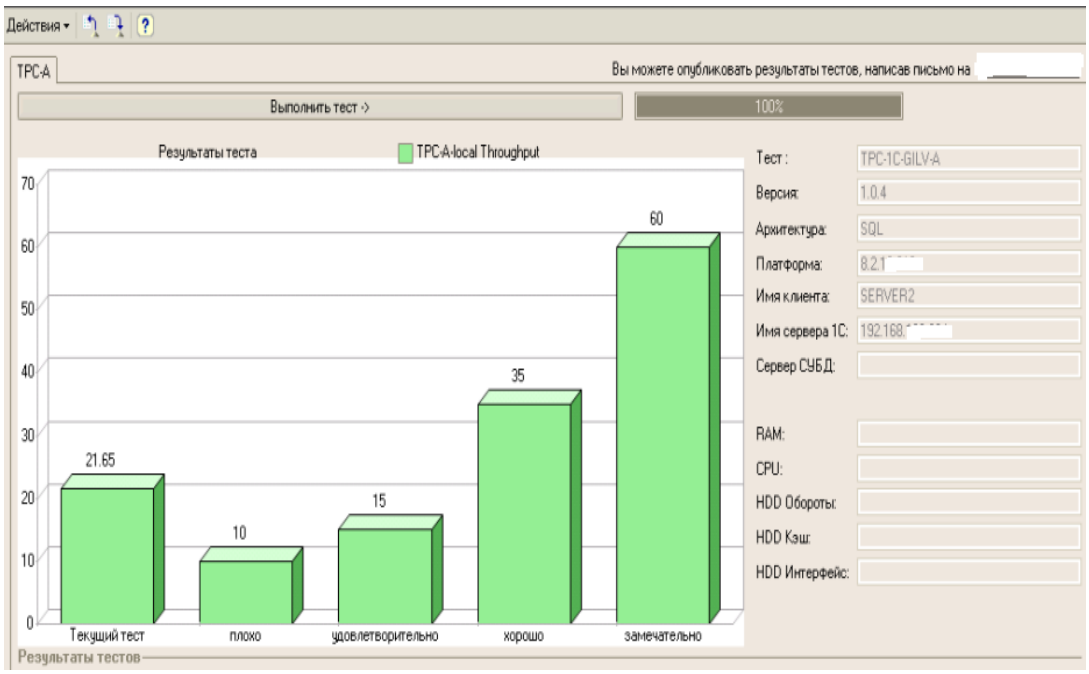

Рисунок 4.10 – Диаграмма результатов тестирования серверов бизнес-приложения 1С8

Зеленый цвет диаграммы в совокупности с дополнительными показателями справа позволяет сделать обобщенную оценку о приемлемой производительности используемых в бизнес-приложении серверов.

Таким образом, на основе предлагаемой методики проведено полнофункциональное тестирование бизнес-приложения 1С8 с помощью специализированных средств автоматизации, что обеспечивает эффективность данного процесса благодаря снижению влияния человеческого фактора.

Выводы к главе 4

1) Для автоматизации процесса тестирования конфигурации бизнес-приложения используется встроенный механизм автоматизации платформы «1С: Предприятие 8.3».

2) Для проведения функционального тестирования конфигурации бизнес-приложения разработаны сценарии, реализованные в виде внешних обработок 1С8.

3) На основе автоматизированного теста TPC-1C выполнено нагрузочное тестирование терминального сервера, сервера приложений 1С (сервер 1с) и СУБД PostgreSQL, которое позволило сделать обобщенную оценку о приемлемой производительности используемых серверов.

4) На основе предлагаемой методики проведено полнофункциональное тестирование бизнес-приложения 1С8 с помощью специализированных средств автоматизации, что обеспечивает эффективность данного процесса благодаря снижению влияния человеческого фактора.

ЗАКЛЮЧЕНИЕ

Целью магистерской диссертации является исследование и разработка методики тестирования платформенных бизнес-приложений, обеспечивающей высокую эффективность данного процесса.

Выполненные в работе научные исследования представлены следующими основными результатами:

1. Произведен анализ существующих видов тестирования программного обеспечения, который подтвердил отсутствие общепринятых рекомендаций по применению конкретных видов тестирования для платформенных бизнес-приложений.

2. Произведен анализ методов и моделей тестирования программного обеспечения, который показал, что для тестирования бизнес-приложений могут использоваться различные методы тестирования и их комбинации. Подтверждена ключевая роль тестирования в жизненном цикле бизнес-приложений. Отмечена важная роль автоматизированного тестирования в сокращении времени внедрения бизнес-приложения и снижении влияния человеческого фактора на результаты тестирования.

3. Произведен анализ существующих методик тестирования платформенных бизнес-приложений, который подтвердил отсутствие универсальной методики тестирования и актуальность темы исследования. Разработана методика тестирования бизнес-приложений, реализованных на платформе «1С: Предприятие 8». Основными направлениями предлагаемой методики являются тестирование конфигурации и серверной части бизнес-приложения.

4. Выполнена апробация и подтверждена возможность применения предложенной методики для повышения эффективности тестирования платформенных бизнес-приложений. В процессе тестирования использованы средства автоматизация тестирования платформы «1С: Предприятие 8».

Таким образом, в работе решена актуальная научно-исследовательская задача разработки методики платформенных бизнес-приложений, обеспечивающей повышение эффективности данного процесса.

Значение диссертационной работы определяется тем, что в ее рамках исследованы возможности повышения эффективности процесса тестирования платформенных бизнес-приложений и предложена методика, обеспечивающая решение данной задачи.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Нормативно-правовые акты

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.

Научная и методическая литература

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.

Электронные ресурсы

8. 1С: Предприятие 8 [Электронный ресурс]. — Режим доступа: http://v8.1c.ru/ (дата обращения: 09.03.2019).

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: 09.03.2019).

10. Загрузки веб-сервисов [Электронный ресурс]. — Режим доступа: https://infostart.ru/public/1014870/?utm_source=subscribe&utm_campaign=week&utm_term=16 (дата обращения: 09.03.2019).

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: 09.03.2019).

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: 09.03.2019).

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: 09.03.2019).

14. Нагрузочный тест TPC-1C [Электронный ресурс]. — Режим доступа: http://www.gilev.ru/tpc1cgilv/ (дата обращения: 09.03.2019).

15. Оценка эффективности автоматизации тестирования [Электронный ресурс]. — Режим доступа: https://www.a1qa.ru/blog/otsenka-effektivnosti-avtomatizatsii-testirovaniya/ (дата обращения: 09.03.2019).

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 09.03.2019).

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 09.03.2019).

18. AQA – система автоматизированного тестирования бизнес-приложений [Электронный ресурс]. — Режим доступа:

https://www.galaktika.by/aqa-cistema-avtomatizirovannogo-testirovaniya-biznes-prilozhenij.html (дата обращения: 09.03.2019).

19. Business Application Testing [Электронный ресурс]. — Режим доступа: https://www.precisetestingsolution.com/business-application-testing.php (дата обращения: 09.03.2019).

20. Functional Testing: A Complete Guide with Types and Example [Электронный ресурс]. — Режим доступа: https://www.softwaretestinghelp.com/guide-to-functional-testing/ (дата обращения: 09.03.2019).

21. Gartner consulting [Электронный ресурс]. — Режим доступа: https://www.gartner.com (дата обращения 09.03.2019).

22. Guru99 [Электронный ресурс]. — Режим доступа: https://www.guru99.com/ (дата обращения: 09.03.2019).

23. How to Write Test Cases: Sample Template with Examples [Электронный ресурс]. — Режим доступа: https://www.guru99.com/test-case.html (дата обращения: 09.03.2019).

24. Software Testing Fundamentals [Электронный ресурс]. — Режим доступа: http://softwaretestingfundamentals.com/ (дата обращения: 09.03.2019).

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: 09.03.2019).

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: 09.03.2019).

Литература на иностранном языке

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, ", ");
```

```
            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
```