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

Кафедра Прикладная математика и информатика
(наименование)

09.04.03 Прикладная информатика
(код и наименование направления подготовки)

Управление корпоративными информационными процессами
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему Исследование и анализ проблем и тенденций развития технологии автоматизации тестирования в информационных системах

Обучающийся А.А. Владимирцева _____
(Инициалы Фамилия) (личная подпись)

Научный руководитель к.т.н., доцент, О.В. Аникина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Оглавление

Введение

Создание методов автоматического тестирования программного обеспечения является актуальной задачей нашего времени. Параметры интерфейсных функций необходимо формулировать в виде объектов различной сложности и структуры. При этом важно, чтобы последовательность вызова интерфейсных функций была адекватна и не содержала ошибок, т.к. в противном случае это приведет к нарушению связей программного интерфейса.

Тема исследования сформулирована как "Исследование и анализ проблем и тенденций развития технологии автоматизации тестирования в информационных системах".

При этом основная проблема исследования - недостаточная эффективность технологии автоматизации тестирования в информационных системах.

Объект исследования: процесс тестирования ПО.

Предмет исследования: методика автоматизации тестирования.

Цель работы - разработка обеспечивающей повышение эффективности методики тестирования ПО с учетом всестороннего анализа проблем и тенденций развития технологии автоматизации тестирования.

Гипотеза исследования - применение предлагаемой методики позволит повысить эффективность процесса тестирования ПО.

В контексте достижения цели и проверки обозначенной гипотезы разумно сформулировать следующие задачи:

- проанализировать известные на сегодня виды тестирования программного обеспечения в контексте проблематики исследования;
- проанализировать методы и модели тестирования программного обеспечения с учетом актуальных требований и стандартов;
- разработать усовершенствованную методику автоматизированного тестирования ПО;

– выполнить апробацию и обосновать применение предложенной усовершенствованной методики компонентного тестирования ПО для повышения эффективности тестирования ПО в ООО Энерговектор.

На защиту выносятся следующие положения:

– методика тестирования ПО, включающая улучшенные алгоритмы автоматизированного тестирования;
– результаты апробации указанной методики тестирования ПО.

Научная новизна данной научно-исследовательской работы заключается в следующих моментах:

– разработан метод автоматизации тестирования ПО с акцентом на автоматизацию процесса выстраивания модели API-интерфейса, расширение и улучшение функционала в рамках спецификаций и с учетом структуры поэтапной генерации тестового кода;
– создан алгоритм для визуализации усовершенствованной модели, направленной на анализ спецификаций интерфейса;
– предложен алгоритм обхода, отвечающий за автоматическое создание тестов в виде цепочки вызовов методов и условий проверки ПО и его функций. Такой предложенный нами алгоритм направлен на повышение качества управления тестовым покрытием разработанной модели;
– предложен и успешно протестирован алгоритм оптимизации тестов, делающий упор на непрерывный анализ тестового набора.

В проведенном нами исследовании рассматриваются различные подходы к тестированию ПО, включая тестирование, независимое от спецификаций, тестирование, основанное на спецификациях, и тестирование аппаратных средств. На основе этих методов была предпринята попытка создать методику автоматизированного тестирования для информационных систем.

В данной работе было предложено уточненное определение комплекса действий компонентного метода, выделены дополнительные критерии компонентного тестирования, обозначены уточненные спецификации компонентного тестирования.

Помимо этого, в данной ВКР было сформулировано уточненное определение совокупности действий метода, описаны актуальные в работе ООО Энерговектор программные составляющие и их мутации (переменные и их присваивание, арифметические выражения, логические отношения, булевские выражения и пр.).

В работе описаны программные конструкции и процесс их тестирования в ООО Энерговектор (операторы циклов, условные и присваивания); уточнены понятия эффективности и полноты тестирования (результативность/полнота тестовых множеств).

На основе полученных данных был проведен сравнительный анализ тестовых методов и разработан улучшенный алгоритм оптимизации автоматизированного тестирования.

Глава 1 Теоретические аспекты тестирования информационных систем

1.1 Анализ и интерпретация научной литературы по теме исследования

Современное развитие программно-технических средств, характеризующееся повышением их сложности и функциональной насыщенности, напрямую связано с широкомасштабным внедрением информационных систем в самые различные сферы человеческой деятельности. Усложнение этих систем, сопровождаемое увеличением их взаимосвязанности и зависимости от разнообразных внешних факторов, неизбежно обуславливает возрастание риска возникновения ошибок, которые могут повлечь за собой серьезные последствия.

В связи с этим возрастает необходимость проведения более тщательной и всесторонней верификации сложных программных комплексов, подразумевающей детальное изучение их корректности, функциональности и надежности на всех этапах разработки и эксплуатации [3]. Данный процесс, включающий в себя использование методов статического и динамического анализа, тестирования на соответствие заданным спецификациям и моделирования различных эксплуатационных сценариев, позволяет минимизировать вероятность появления сбоев или некорректного поведения системы [7], [8], [9].

Кроме того, учитывая стремительное расширение областей применения подобных технологий, таких как здравоохранение, транспорт или банковская сфера, недостаточная проверка программных средств может привести к значительным финансовым потерям, нарушению безопасности данных или даже к угрозе жизни и здоровью людей. Следовательно, верификация, выполняющая роль основного инструмента для оценки соответствия системы

ее целям и задачам, становится важнейшим этапом обеспечения их качества и надежности [10], [12].

Программные средства в информационных системах, являясь многоуровневыми объектами, характеризуются высокой сложностью, что затрудняет их анализ и иногда делает его практически невозможным. Такая сложность обусловлена, в том числе, необходимостью учитывать множество факторов, среди которых — степень сложности самой программы, ограниченные ресурсы времени и многообразие входных данных, требующих проверки для подтверждения корректности выходных результатов [16], [18]. В этих условиях процесс верификации требует внедрения автоматизированных систем тестирования, что позволяет значительно упростить и ускорить процесс обеспечения надежности и эффективности программных комплексов [24], [25], [26].

Тема исследования проблем автоматизации тестирования хорошо освещена западными учеными и программистами такими, как Ли Коплэнд [50], Рон Паттон, Джеральд Вайнберг [65], Джеймс Уиттакер [66], [67], Г. Майерс, Т.Баджетт, К.Сандлер, Д. Макгрегор, Д.Сайкс, Д. Мосли, Дж. Поузи и др., которые сосредотачивают свое внимание на практической стороне вопроса.

Западные исследователи, такие как Ли Коплэнд [50], Рон Паттон [62] и Джеральд Вайнберг [65], в значительной степени сосредоточены на практических аспектах автоматизации тестирования, что нашло отражение в многочисленных работах. Однако, несмотря на их вклад в развитие данной области, программные системы остаются слишком сложными объектами для ручной проверки, что делает необходимым глубокое изучение теоретических основ автоматизации тестирования [28], [29] [31], [32].

Таким образом, чтобы найти ответ на большинство проблем автоматизации тестирования необходимо углубиться в теоретическую базу данного вопроса. Развитием теоретической методологии автоматизации тестирования в разное время занимались такие ученые как С.В.Эмблер [49],

П.Садаладж, Дастин Эльфриде [51], [52], В.Гарузи [53], Л.Рапанотти, К.Джеквони [56], Дж. Джаки [57], М.Винс, К.Луттерот [59], С.Родзин [63], Л.Родзина, Р. Скотт Барбер [64], Бек Кент, Адам Колава [19] и другие.

С развитием методологии тестирования особое внимание уделяется использованию моделей ПО. В этом контексте значимыми являются технологии, такие как UniTESK, разработанная в Институте системного программирования РАН, и AsmL от Microsoft Research, которые показали высокую эффективность при решении задач автоматизации тестирования программных комплексов [35], [36], [37].

Тестирование остаётся центральным элементом обеспечения надёжности программных систем, что подчёркивается и современной ситуацией на рынке труда. В развитых странах наблюдается значительный спрос на специалистов в области тестирования, превышающий спрос на разработчиков программного обеспечения. Это свидетельствует о важности и востребованности дальнейших исследований и разработок в сфере автоматизации тестирования, направленных на повышение качества и надёжности программных продуктов [39], [41], [42], [43].

Проблемами потребностей и инноваций в сфере автоматизации тестирования занимаются такие ученые, как Э.Ли [58], Б.Мейер [60], Дж. Уиттакер [66], [67], А.Заррад [69], Дж.Вайнберг [65], С. Макконнелл, Линда Хейс, А.М. Голубев [11], П.Д.Дробинцев [14], Д.В.Кадашев [17], В.А.Лазарев, А.С.Мартюков, Р.Савин [40] и другие.

Тестирование программного обеспечения не может дать исчерпывающей гарантии корректности программы. Основополагающим моментом тестирования является его отношение к верификации, которая представляет собой предельный случай идеальной проверки корректности программ. Верификация, как эталон, становится основой для оценки эффективности тестирования, так как она определяет рамки идеальной программной корректности, к которой тестирование только приближается [45], [47], [55], [68].

Вопросы подходов в автоматизации тестирования достаточно хорошо освещены в работах Дж.Р.Монсмы [61], Х.Жу [70], П.А.Холла [54], Дж.Х.Мэя, Р.Блэка [6], А.С.Артюховой [2], С.А.Баркалова, Т.В.Азарновой, П.В.Полухина [4], А.Л.Забровского [15], С.Куликова [20], В.Кулянина [22], А.Петренко, А.Косачева, И.Бурдонова, В.М.Курейчика, С.И.Родзина [23], А.В. Мищенко [33], А.Ф. Шайхутдиновой [48], Д.М.Плодухина [38] и других.

Отсюда вытекает необходимость разработки критериев, которые позволяли бы не только сравнивать различные методы тестирования, но и оценивать их по такому показателю, как "мощность" теста, то есть его способность обнаруживать ошибки и повышать уровень доверия к программам. При этом важно понимать, что программы, в отличие от аппаратных систем, могут быть математически корректными или некорректными, но надёжность программы не сводится лишь к факту отсутствия отказов во время тестирования, что усложняет задачу построения универсальных методов тестирования.

Существенной проблемой остаётся и тот факт, что, хотя успешное прохождение теста укрепляет доверие к программе, оно не устраняет вероятность наличия скрытых ошибок. Следовательно, в теории тестирования возникает запрос на развитие методов, которые бы, с одной стороны, приближались к верификации программ, а с другой — упрощали и автоматизировали процесс тестирования, сохраняя его практическую применимость.

Рассматривая тестирование как приближение к верификации, становится очевидным, что теоретическое обоснование процесса тестирования должно учитывать и вопрос завершимости программ. Таким образом, теория тестирования программного обеспечения находит своё развитие на пересечении вопросов корректности, надёжности и автоматизации, что требует дальнейшего теоретического и практического осмысления.

1.2 Теоретические основы и проблемы тестирования в контексте обеспечения надежности информационных систем

Теоретическая база тестирования с позиции обеспечения надежности играет ключевую роль в современном мире, где информационные технологии становятся основой большинства процессов, протекающих как в государственных, так и в частных организациях. Надежность информационных систем, заключающаяся в их способности корректно функционировать в условиях внешних и внутренних воздействий, напрямую зависит от качества их тестирования. Процесс тестирования позволяет выявить уязвимости, ошибки в программном коде и возможные сбои, которые могут привести к катастрофическим последствиям, начиная от потери данных до нарушения целостности информации или отказа системы. Теоретические подходы к тестированию, включающие методы статического и динамического анализа, позволяют более точно и своевременно предвидеть возможные проблемы в функционировании систем, что обеспечивает их устойчивость к различным нагрузкам и сбоям. Более того, без тщательной проверки надежности невозможно гарантировать безопасность системы, которая становится особенно важной в условиях постоянных кибератак и угроз несанкционированного доступа [5].

Проблемы тестирования зачастую связаны с тем, что многие существующие методики не охватывают весь спектр потенциальных рисков, либо требуют значительных ресурсов для их реализации. Сложность современных информационных систем, имеющих множество взаимосвязанных компонентов, делает их тестирование трудоемким процессом, который нуждается в постоянном обновлении и адаптации к новым вызовам. Важность правильного тестирования заключается в том, что даже незначительные ошибки могут иметь масштабные последствия, приводящие к неработоспособности систем, что делает задачу обеспечения надежности особенно актуальной в цифровом мире.

Тестирование программного обеспечения, являющееся неотъемлемой частью процесса верификации, представляет собой сложный и многогранный процесс, направленный на выявление и устранение ошибок, возникающих в программных продуктах. Верификация, будучи одной из ключевых методологий, предполагает анализ программных объектов с целью сопоставления их текущего состояния с требованиями, предъявляемыми заказчиком и спецификацией. Данная деятельность, направленная на проверку правильности работы программы и соответствие её функционала ожиданиям, осуществляется на протяжении всего жизненного цикла разработки программного обеспечения.

Одной из главных задач тестирования, выполнение которой критически важно для всех участников проекта, является предоставление гарантий того, что программный продукт удовлетворяет требованиям, изложенным в технической документации. При этом, наряду с выявлением соответствия программы требованиям, не менее значимым аспектом тестирования становится обнаружение ситуаций, когда поведение программы оказывается неверным, отклоняющимся от спецификации или вовсе не отвечающим ожиданиям пользователей. Именно на этом этапе возникает необходимость в своевременном выявлении сбоев и слабых мест, что может привести к значительным временным и финансовым затратам, составляющим существенную часть общего бюджета разработки программного обеспечения.

В целях минимизации затрат времени и ресурсов на выявление дефектов, характерных для ручного тестирования, разработчики всё чаще прибегают к автоматизированным подходам, которые, благодаря использованию специализированных программных средств, позволяют ускорить процесс контроля качества. Автоматизированное тестирование, начало активного развития которого приходится на 1980-е годы, несмотря на более ранние попытки внедрения подобных практик, является важным инструментом в обеспечении надёжности программного продукта. Данный

метод предполагает использование различных подходов, среди которых можно выделить тестирование на уровне кода и тестирование пользовательского интерфейса.

Первый из этих методов, известный как тестирование «белого» ящика, основывается на анализе внутренней структуры программного кода. Тестировщики, обычно являющиеся разработчиками, принимающими участие в создании кода, проверяют, насколько корректно выполняются функции программы на низком уровне, что позволяет выявить ошибки на этапе тестирования по модулям. Модульное тестирование призвано проверить систему на предмет корректности отдельных компонентов, что в свою очередь сокращает время и ресурсозатраты на весь процесс в целом. НАйденные таким образом ошибки можно легко обнаружить и исправить.

Так называемое "черноящичное" тестирование предполагает тестирование с учетом спецификаций. В данном подходе акцент делается на высокоуровневую оценку поведения системы без анализа внутренней структуры кода. Использование методов черного ящика, несмотря на свою эффективность в проверке соответствия ПО ожиданиям, порождает ряд сложностей, особенно в условиях современного agile-подхода к разработке [6].

Одной из наиболее значительных проблем автоматизированного тестирования является его трудоемкость. Хотя автоматизация снижает рутинную нагрузку на тестировщиков, позволяя ускорить процесс выполнения тестов, разработка и регулярное обновление тестов требуют значительных временных ресурсов. Аналогичный вопрос обновления тестов возникает в результате реорганизации кода, когда с учетом изменений требуется обновление юнит-тестов, что на практике может занимать столько же времени, сколько и внесение изменений в основной код программы.

Актуальность автоматических тестов становится проблемой для специалистов в связи с частыми изменениями пользовательских интерфейсов, требующими постоянного переписывания тестов, что ведёт к

увеличению затрат на разработку продукта. Автоматизация тестирования, дополняя, но не заменяя ручное тестирование, оказывается эффективной, особенно в области регрессионного тестирования. Следует учитывать, что правильно организованное сочетание автоматического и ручного тестирования способствует повышению качества и стабильности продукта, что особенно важно в условиях быстрой разработки и высокой конкуренции. Такой подход позволяет сосредоточить усилия разработчиков на более сложных задачах, эффективно снижая затраты и ускоряя процессы тестирования.

GUI-автоматизация является одной из ключевых технологий в области тестирования программного обеспечения. Основной причиной её популярности служат два аспекта: во-первых, тестирование происходит в условиях, максимально приближенных к реальному использованию программы; во-вторых, тесты могут выполняться автоматически и непрерывно, что значительно повышает производительность и исключает необходимость в постоянном участии человека [11].

Выделяют четыре поколения данной технологии.

Первое поколение, представленное утилитами записи и воспроизведения (capture/playback tools), характеризуется возможностью автоматической фиксации действий тестировщика в процессе ручного тестирования. Эти инструменты, освобождая от необходимости повторных ручных операций, позволяют запускать тесты без вмешательства пользователя на протяжении длительного времени. Однако основным недостатком данного метода является его слабая адаптивность: малейшее изменение в интерфейсе приложения требует полной перезаписи тестов, что, в свою очередь, значительно снижает масштабируемость и эффективность данного подхода в условиях динамически развивающегося ПО.

Следующим этапом развития стала методика написания сценариев (scripting), предполагающая использование специализированных языков программирования для создания автоматических тестов. Хотя данный подход

позволяет избежать ряда недостатков, свойственных предыдущему поколению инструментов, разработка сценариев требует участия высококвалифицированных программистов, работающих в тесной координации с тестировщиками. Кроме того, такие сценарии, несмотря на свою гибкость, в основном направлены на тестирование графического интерфейса и редко могут быть интегрированы в более широкие процессы тестирования, что ограничивает их универсальность. Постоянное обновление тестируемого приложения требует параллельного обновления скриптов, что усложняет поддержание целостной библиотеки тестов.

Третье поколение, известное как управляемое данными тестирование (Data-driven testing), значительно расширяет возможности автоматизации. Основное преимущество данного метода заключается в том, что тестирование основывается на переменных, которые могут изменяться без необходимости переписывания основного кода тестов, что значительно повышает их гибкость и масштабируемость. Однако это также требует тщательной организации данных и структурирования тестов в рамках специализированного Фреймворка.

Наконец, четвёртое поколение представлено тестированием по ключевым словам (Keyword-based testing), позволяющим упростить процесс автоматизации благодаря разделению разработки тестов на этапы планирования и реализации. Суть этого метода заключается в том, что тестовые кейсы создаются на основе набора ключевых слов, каждое из которых представляет определённое действие или условие. Реализация действий осуществляется Фреймворком, что освобождает тестировщиков от необходимости программирования, делая процесс создания тестов доступным для специалистов с минимальными техническими навыками.

Таким образом, эволюция инструментов и методов GUI-автоматизации направлена на повышение гибкости, масштабируемости и доступности тестирования, несмотря на постоянное усложнение технологий и требований к качеству программного обеспечения.

GUI-автоматизация сталкивается с рядом проблем, одной из которых является отсутствие универсального метода, так как каждый проект обладает уникальными характеристиками. В рамках гибкого тестирования, ориентированного на бизнес-задачи и требования заказчика, автоматизация играет ключевую роль, особенно при использовании подхода разработки через тестирование (TDD). Данный метод, предложенный Кентом Беком, предполагает цикличное создание тестов, написание кода для их прохождения и последующий рефакторинг. Несмотря на активное развитие таких инструментов, как JUnit и его аналоги (например, NUnit), TDD имеет ряд ограничений: сложности с автоматизацией тестирования графического интерфейса и объектов распределения, неактуальность данной методики для работ с базами данных, компиляторами или сторонним кодом[13].

GRID-технологии, активно развивающиеся в последнее время благодаря усилиям специалистов, таких как Д. В. Кадашев и А. А. Кузнецов [17, с. 20], представляют собой перспективный подход к распределённым вычислениям, при котором несколько компьютеров, объединённые в сеть, могут действовать как единый вычислительный ресурс. Такой подход особенно полезен для решения сложных задач, требующих значительных вычислительных мощностей, что достигается за счёт объединения ресурсов с использованием специализированного программного обеспечения.

Одной из ключевых областей применения таких технологий становится разработка систем распределённого модульного тестирования.

Сама по себе GRID-технология строится на принципе объединения разнородных вычислительных ресурсов в одну структуру, состоящую из нескольких управляющих серверов, что позволяет распределять задачи между узлами сети. Этот механизм открывает новые горизонты для модульного тестирования (unit-тестирования), обеспечивая возможность одновременного выполнения множества тестов на различных машинах.

Тесты в данном случае выступают программными компонентами, проверяющими корректность работы других компонентов системы, что

позволяет своевременно выявлять ошибки при внесении изменений в программное обеспечение.

Однако, несмотря на потенциал, GRID-системы имеют ряд ограничений. Одним из таких недостатков является их неэффективность в условиях обработки запросов в реальном времени, где предпочтительнее использовать вертикально масштабируемые OLTP-системы. Кроме того, текущие инструменты для разработки программного обеспечения с применением GRID-технологий всё ещё нуждаются в совершенствовании, особенно в области интеграции с веб-сервисами.

Выводы по первой главе

Таким образом, проведенное исследование подтверждает важность комплексного подхода к тестированию информационных систем, сочетания ручных и автоматизированных методов, а также активного применения новых технологий для повышения надежности и качества программного обеспечения.

Глава 2 Анализ практических методов тестирования ПО в информационных системах

2.1 Методы автоматизации тестирования API

До недавнего времени отдел контроля качества и разработчики использовали основные платформы автоматизации тестирования с открытым исходным кодом, например, Selenium, Appium и другие решения для написания сценариев на основе кода. У них есть лучшие практики, большие базы знаний и документация, но есть и некоторые недостатки:

Они требуют сильных навыков программирования на Java, JavaScript или другом языке.

Приложение меняется, оно кажется нестабильным

Они используют фреймворки, которые тесно интегрированы со средой разработки, и инженеры по тестированию (например, IntelliJ, Eclipse и т. д.)

Учитывая вышесказанное, команды контроля качества ищут более простые и стабильные решения, которые позволяют сократить время на оценку и обратную связь. Для многих из них ответом является автоматизация тестирования на основе машинного обучения [20], [21], [22], [14].

Автоматизация тестирования API представляет собой важный аспект разработки и эксплуатации современных информационных систем, поскольку API (Application Programming Interface) играет ключевую роль в интеграции различных компонентов и сервисов. В условиях стремительно увеличивающейся сложности программных систем, ручное тестирование API становится не только трудоемким, но и подверженным человеческим ошибкам, что снижает общую надёжность конечного продукта. Введение автоматизированных методов тестирования API позволяет значительно повысить эффективность тестирования и минимизировать возможные риски, связанные с несовместимостью или неправильным взаимодействием между компонентами системы.

На современном этапе развития промышленной разработки программного обеспечения необходимость систематизированного тестирования уже не подлежит сомнению, поскольку сложность и масштаб современных программных систем требуют высокого уровня надежности и качества. Традиционные методы ручного тестирования, основанные на взаимодействии с графическим интерфейсом, в значительной степени устарели, так как не способны охватить все возможные сценарии и состояния сложных программных решений, что может привести к недооценке потенциальных проблем и дефектов. В условиях, когда устранение ошибок на поздних этапах разработки обходится в десятки раз дороже, нежели на стадии проектирования, важность эффективного тестирования возрастает многократно [15], [17], [27], [30].

Пренебрежение тестированием API, несмотря на его критическую важность для успешной работы многих современных информационных систем, может привести к значительным сбоям в работе приложений, что особенно опасно в условиях высоких нагрузок и требований к производительности. Автоматизация тестирования API, таким образом, становится важнейшим инструментом для обеспечения не только функциональной корректности программного обеспечения, но и его масштабируемости, надежности и стабильности на различных этапах жизненного цикла продукта.

Систематическое и автоматизированное тестирование API должно рассматриваться как один из ключевых методов повышения качества программных систем, особенно в тех случаях, когда графический интерфейс не играет значительной роли или вовсе отсутствует. Реализуя связь между различными программными компонентами, API позволяет одним программам обращаться к функционалу других без необходимости раскрывать подробности своей внутренней структуры. Таким образом, сокрытие деталей реализации упрощает процесс интеграции и повышает безопасность взаимодействий, оставляя пользователей и разработчиков в

неведении о внутреннем устройстве используемых методов, тем самым поддерживая стабильность и надёжность системы.

Тестирование API представляет собой задачу, требующую учета ряда специфических особенностей, связанных с отсутствием визуальных средств взаимодействия с методами интерфейса. В отличие от тестирования пользовательских интерфейсов, где есть возможность напрямую наблюдать результаты взаимодействий, в случае с API необходимо разрабатывать специальную оболочку, которая будет имитировать выполнение запросов и принимать результаты. Тестировщику необходимо с осторожностью выбирать ограниченный набор значений, которые с наибольшей вероятностью могут выявить дефекты в работе метода, оптимизируя при этом процесс тестирования.

Кроме того, сложность тестирования увеличивается, если функциональность метода зависит не только от отдельных параметров, но и от их сочетаний. Комбинации значений различных параметров могут приводить к неожиданным результатам, что делает необходимым проведение углубленного анализа для выбора критических комбинаций, способных выявить потенциальные проблемы.

Как видим, тестирование API, будучи процессом, включающим множество аспектов, требует точного планирования, оптимизации набора тестов и разработки специфических инструментов, позволяющих с минимальными затратами выявить как можно больше дефектов системы.

Тестирование интерфейсов программирования приложений (API) является важной и сложной задачей, особенно когда методы API требуют использования данных сложных типов в качестве входных параметров. Одной из ключевых проблем, связанных с тестированием API, является необходимость вручную формировать последовательности вызовов методов, что существенно усложняет процесс и увеличивает затраты времени. Каждая цепочка вызовов должна быть корректно выстроена, чтобы промежуточные данные, полученные на одном этапе, могли быть использованы на

следующем. Модульное тестирование, являясь одним из наиболее популярных подходов, фокусируется на автоматизации лишь этапа выполнения тестов, оставляя процесс написания последовательностей вызовов и проверки результатов на ответственности разработчика [19], [44], [46].

Прогресс в области автоматизации тестирования API был достигнут благодаря внедрению методов моделирования программного обеспечения, что позволило унифицировать описание структуры и функциональных требований к системам, открывая возможность их формального анализа. Работы таких исследователей, как Б. Бейзер, Х. Робинсон и Дж. Уиттакер [66], [67], подробно освещают применение моделей ПО в тестировании. Среди наиболее успешных примеров использования этих идей можно отметить технологии UniTESK и AsmL, разработанные соответственно Институтом системного программирования РАН и Microsoft Research, а также технологии Postman, инструменты которого заточены специально под разработку и тестирование API (см. рис. 1).

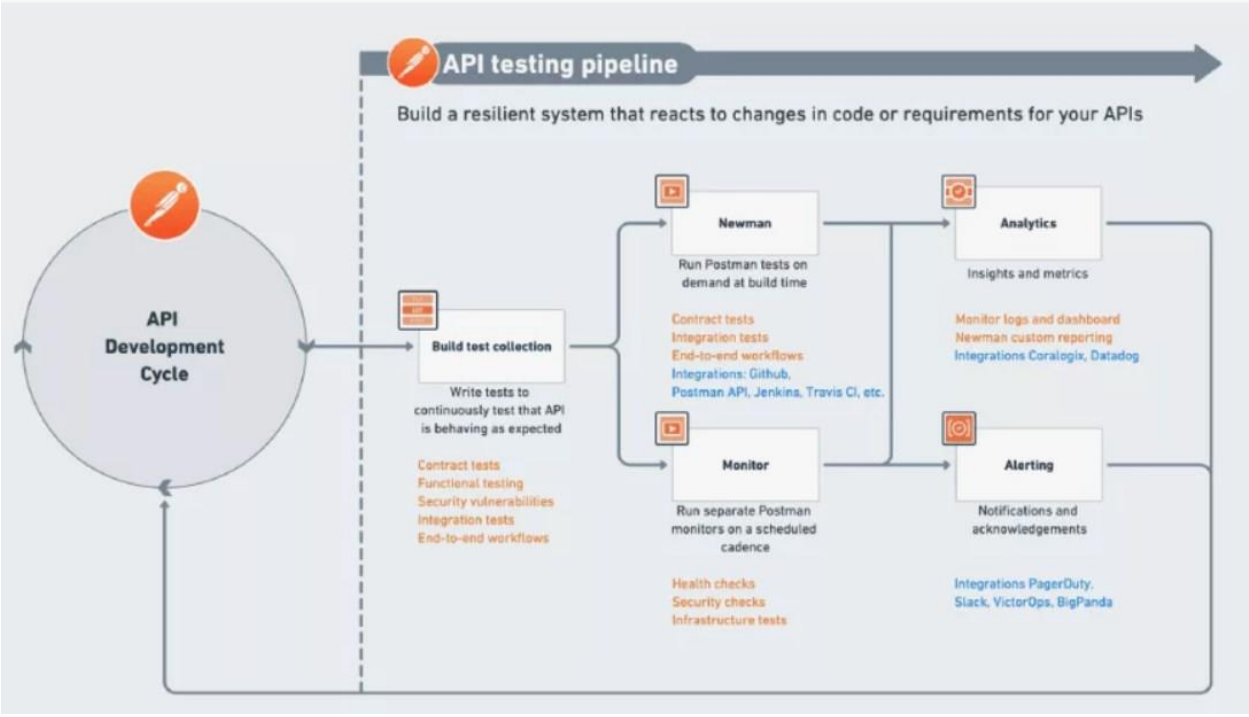

Рисунок 1 - Тестирование API в Postman

Таким образом, дальнейшие исследования должны быть направлены на разработку методов, которые, анализируя спецификацию интерфейса, будут автоматизировать процесс создания тестов, одновременно гарантируя корректность взаимодействия различных компонентов API, что существенно повысит эффективность тестирования и качество получаемых результатов.

Подводя итоги анализа методов автоматизации тестирования API, можно сделать несколько важных выводов, которые отражают текущее состояние и перспективы этой области. Автоматизация тестирования API становится необходимым элементом процесса разработки программного обеспечения, так как она значительно повышает качество и стабильность информационных систем. Этот подход не только снижает вероятность человеческих ошибок, но и ускоряет тестирование за счёт выполнения большого количества сценариев взаимодействия, чего технически невозможно достичь, используя метод ручного тестирования.

Внедрение моделирования программного обеспечения, выполняющее ключевую роль в автоматизации тестирования, представляет собой значительный шаг к стандартизации описания структуры и функциональных требований к системам, что особенно важно в условиях растущей сложности современных информационных технологий. Такой подход, обеспечивающий возможность более точного и формализованного представления спецификаций, позволяет не только ускорить процесс тестирования, но и снизить вероятность ошибок, возникающих вследствие человеческого фактора.

Однако, несмотря на очевидные преимущества, процесс моделирования сталкивается с рядом существенных трудностей, вызванных трудоёмкостью создания моделей, тестовых оракулов и медиаторов, требующих высокой квалификации специалистов. Эти элементы, играющие роль связующих звеньев между тестируемой системой и инструментами

проверки, нуждаются в дальнейшем совершенствовании, чтобы уменьшить их сложность и повысить степень автоматизации.

Одной из приоритетных задач, стоящих перед исследователями и разработчиками, становится создание методов, позволяющих автоматизировать процесс генерации тестов на основе спецификаций интерфейсов API. Достижение этой цели предполагает использование формальных методов анализа и преобразования данных, исключающих необходимость непосредственного вмешательства человека, что значительно упрощает процесс тестирования и повышает его эффективность.

Автоматизация тестирования API, будучи сложным и многоэтапным процессом, обладает колоссальным потенциалом для повышения качества и надёжности программных систем, особенно учитывая современные требования к их масштабируемости, устойчивости и производительности. Успешная реализация таких методов обеспечит не только снижение затрат на тестирование, но и существенное повышение доверия к разрабатываемым информационным системам, играющим критическую роль в различных сферах деятельности.

2.2 Подходы оптимизации бизнес-процессов тестирования (на примере бизнес-процессов ООО Энерговектор)

Подходы к внедрению изменений требуют тщательного планирования и последовательности действий, чтобы избежать сбоев и негативных последствий. Одним из эффективных методов является цикл Деминга (PDCA), который состоит из четырёх этапов (рис. 2).

Рисунок 2 - Этапы цикла Деминга (PDCA)

Еще один эффективный подход — это методология IDEAL, представляющая собой развитие цикла Деминга, но с более детализированной структурой.

I — Identify (Идентификация). На этом этапе происходит выявление проблемы, требующей решения.

D — Define (Определение). Здесь определяется сама проблема и желаемый итог. Каков результат, которого мы хотим достичь?

E — Explore (Исследование). В данном пункте рассматриваются варианты решения, осуществляется их анализ и прогнозирование последствий.

A — Act (Действие). Выбор метода разрешения проблемы и его реализация происходят на этом этапе.

L — Look back (Обратная связь). Завершающий этап цикла включает в себя ретроспективу процесса и принятие решений. Если реализованный метод оказался эффективным, то оставляем его. Если нет — повторяем цикл.

Завершающим подходом в данной категории является *Кайдзен*, который опирается на японскую философию постоянного улучшения всех процессов. Основу этой философии составляют три ключевых принципа: управление рабочим местом, устранение неоправданных потерь, стандартизация процессов.

Рассмотрим модели улучшения процесса тестирования (на примере ООО «Энерговектор»).

Существует четыре категории подходов к совершенствованию процессов (табл. 1).

Таблица 1 - Категории подходов к совершенствованию процессов

Категория подхода	Описание
Подходы, основанные на моделях	Использование общих или специализированных моделей для улучшения процессов тестирования и достижения зрелости команды.
Аналитические подходы	Реактивные методологии, направленные на установление причинно-следственных связей для устранения и предотвращения ошибок.
Подходы, основанные на метриках	Использование количественных показателей для оценки и улучшения процессов тестирования.
Гибридные модели	Комбинация различных подходов для достижения оптимальных результатов в конкретном контексте организации.

Подходы, основанные на моделях, включают в себя общие модели управления или улучшения, такие как CMMI, и узкоспециализированные модели, например TMAP, TPI Next и TMMi. Общие модели охватывают все сферы разработки программного обеспечения и требуют внесения изменений на каждом этапе процесса. Они характеризуются комплексностью и последовательностью, что делает их подходящими для длительных проектов, больших команд и организаций, стремящихся к сертификации и стандартизации процессов. Однако такие модели не подходят для короткосрочных проектов с небольшими командами, требуют значительного вовлечения руководства и могут быть неэффективны для неповторяющихся процессов. Узкоспециализированные модели сосредоточены на улучшении тестирования или качества продукта и помогают достичь зрелости команды

через последовательные уровни развития. Они эффективны для фокусированного улучшения конкретных аспектов тестирования и профессионального роста команды, но могут иметь ограничения в применении и требовать адаптации под специфику организации (табл. 2).

Таблица 2 - Оптимизация тестирования на основе моделей

Тип модели	Примеры	Описание	Преимущества	Недостатки
Общие модели управления или улучшения	CMMI	Охватывают все сферы разработки ПО, требуют внесения изменений на каждом этапе.	- Комплексность и последовательность - Подходят для длительных проектов - Эффективны в больших командах - Способствуют сертификации - Интеграция с существующими процессами	- Не подходят для короткосрочных проектов с небольшой командой - Требуют вовлечения руководства - Неэффективны для неповторяющихся процессов
Узкоспециализированные модели	TMAP, TPI Next, TMMi	Нацелены на улучшение тестирования или качества продукта и достижение зрелости команды через последовательные уровни зрелости.	- Фокус на конкретных аспектах тестирования - Помогают в достижении зрелости команды	- Могут быть ограничены в применении - Возможна необходимость адаптации к специфике организации

Выбор между этими подходами зависит от контекста проекта и организации: для крупных и длительных проектов с потребностью в стандартизации и сертификации больше подходят общие модели, тогда как для целенаправленного улучшения процессов тестирования и развития команды эффективнее использовать узкоспециализированные модели. При этом необходимо учитывать размер и продолжительность проекта, состав и зрелость команды, степень вовлеченности руководства и характер процессов. Организация должна тщательно проанализировать свои потребности, ресурсы и цели, чтобы выбрать наиболее подходящую модель или комбинацию моделей для оптимизации процессов тестирования.

Модели предыдущей категории относятся к проактивным подходам, в то время как аналитические представляют собой реактивные методологии управления, что не умаляет их эффективности [33], [34], [50].

Аналитические подходы представляют собой реактивные методологии, направленные на установление причинно-следственных связей для устранения и предотвращения ошибок. Эти методы особенно эффективны, когда необходимо определить коренную причину возникшей проблемы с целью её устранения и предотвращения повторения в будущем. Аналитические подходы отлично сочетаются с другими методологиями, усиливая их эффективность. Ключевыми методами в рамках аналитических подходов являются Диаграмма Исикавы (Fishbone Diagram), метод "5 Почему" и GQM (Goal/Question/Metric).

Диаграмма Исикавы, также известная как "рыбья кость", служит визуальным инструментом для выявления потенциальных причин проблемы. Она начинается с определения основной проблемы и последующего разветвления на различные процессные области или факторы, которые могли на неё повлиять. Этот метод позволяет получить целостное представление о проблеме, выявить неочевидные взаимосвязи и углубить понимание процессов, что может привести к обнаружению не только непосредственной

причины ошибки, но и слабых мест в процессе, предотвращая будущие инциденты.

Метод "5 Почему" отличается своей простотой и эффективностью. Он предполагает последовательное задавание вопроса "Почему?" к каждому полученному ответу относительно проблемы до тех пор, пока не будет выявлена коренная причина. Обычно достаточно пяти итераций, чтобы достичь исчерпывающего объяснения. Этот метод хорошо сочетается с Диаграммой Исикавы, позволяя более точно определить области влияния и факторы, приведшие к возникновению ошибки.

GQM (Goal/Question/Metric) представляет собой трехуровневый подход к анализу и улучшению процессов. На концептуальном уровне формулируется ясная цель или задача, отражающая ожидаемый результат и преимущества от её достижения. Операционный уровень включает разработку вопросов, направленных на определение методов и стратегий достижения цели. На количественном уровне подбираются метрики, которые позволяют дать измеримые и конкретные ответы на поставленные вопросы. Разработанный NASA, этот метод подходит как для планирования точечных изменений, так и для внедрения масштабных инноваций.

Таблица 3 - Аналитические подходы

Метод	Описание	Преимущества
Диаграмма Исикавы (Fishbone Diagram)	Визуальный инструмент для выявления причинно-следственных связей, влияющих на возникновение проблемы. Начинается с определения проблемы и разветвляется на возможные причины.	- Позволяет увидеть проблему в целом - Выявляет неочевидные взаимосвязи - Помогает обнаружить слабые места процесса

Продолжение таблицы 3

Метод	Описание	Преимущества
Метод "5 Почему"	Последовательное задавание вопроса "Почему?" для выявления коренной причины проблемы. Обычно достаточно пяти итераций для достижения результата.	- Простота и эффективность - Глубокое понимание проблемы - Хорошо сочетается с другими методами
GQM (Goal/Question/Metric)	Трехуровневый метод анализа: Цель (концептуальный уровень), Вопросы (операционный уровень), Метрики (количественный уровень).	- Четкое формулирование целей - Определение стратегии достижения - Измеримые результаты

В целом, аналитические подходы ценны своей способностью глубоко анализировать проблемы и выявлять их коренные причины. Они особенно полезны в ситуациях, когда понимание фундаментальных вопросов критично для улучшения процессов. Благодаря своей гибкости, эти методы могут эффективно комбинироваться с другими подходами, усиливая общую эффективность процессов тестирования и позволяя организациям более оперативно и точно реагировать на возникающие проблемы.

Улучшение процессов тестирования на основе метрик (на примере ООО Энерговектор). Основные правила использования метрик:

- терпение. Сбор метрик необходимо осуществлять заранее;
- релевантность. Это правило подразумевает грамотный подбор метрик и их оптимальное количество;
- точность. Следует использовать только точные и взаимосвязанные метрики, которые смогут в совокупности обрисовать общую картину.

Метрики, рекомендованные к использованию ООО Энерговектор в целях оптимизации процессов тестирования представлены на схеме (рис. 3).

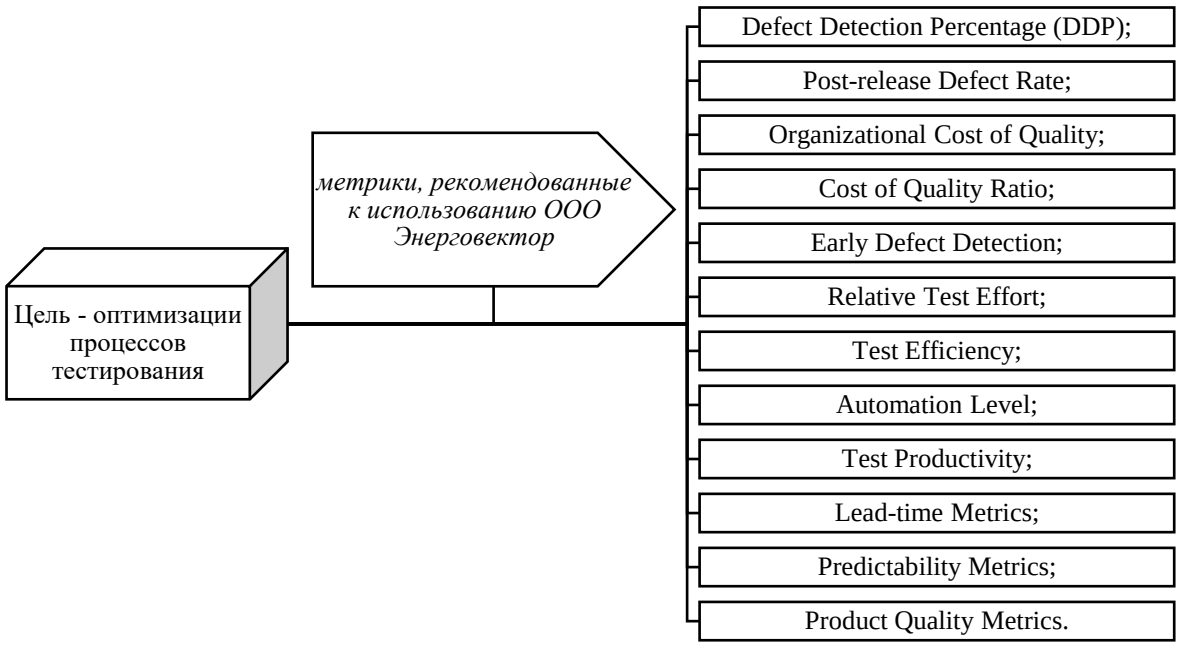

Рисунок 3 - Метрики, рекомендованные к использованию ООО Энерговектор в целях оптимизации процессов тестирования

Рекомендуемые метрики для ООО "Энерговектор" представляют собой всесторонний набор инструментов для оценки и оптимизации процессов тестирования и качества продукта. Метрики, такие как Defect Detection Percentage (DDP) и Early Defect Detection, позволяют определить эффективность выявления дефектов до релиза, что способствует снижению рисков и затрат, связанных с исправлением ошибок на поздних стадиях разработки. Post-release Defect Rate помогает оценить качество продукта после его выпуска, предоставляя ценные данные для дальнейшего улучшения процессов тестирования и разработки.

Экономические показатели, такие как Organizational Cost of Quality и Cost of Quality Ratio, дают возможность анализировать затраты на обеспечение качества и соотносить их с общими расходами на проект. Метрика Relative Test Effort помогает понять долю усилий, затраченных на

тестирование, в контексте всего проекта, что важно для эффективного распределения ресурсов.

Test Efficiency и Test Productivity предоставляют информацию об эффективности и продуктивности тестовых процессов, позволяя выявить области для улучшения и повысить общую результативность команды. Automation Level отражает степень автоматизации тестирования, что является ключевым фактором в ускорении процессов и снижении человеческого фактора в ошибках.

Метрики Lead-time Metrics и Predictability Metrics позволяют оценить временные аспекты разработки и тестирования, такие как продолжительность циклов и точность планирования. Наконец, Product Quality Metrics дают общее представление о качестве продукта, включая такие аспекты, как стабильность и производительность, что напрямую влияет на удовлетворенность пользователей и репутацию компании [51], [59].

В совокупности использование этих метрик позволит ООО "Энерговектор" получить глубокое понимание эффективности своих процессов тестирования и качества продукта. Это создаст основу для принятия информированных решений по оптимизации процессов, повышению качества и эффективности, что в конечном итоге приведет к улучшению конкурентоспособности компании и удовлетворенности клиентов.

Гибридный подход к оптимизации тестирования. Подход к улучшению тестирования и управлению процессами не может быть единообразным. Закон Парето утверждает, что «20 % усилий приносят 80 % результата, тогда как оставшиеся 80 % усилий обеспечивают лишь 20 % результата». Если правильно расставить приоритеты задач, аккуратно выбрать методы их решения и выполнить первые 20 % работы, то отличные результаты не заставят себя ждать. Рекомендуемая дата для проверки показателей оптимизации процессов тестирования в ООО «Энерговектор» — 01.01.2026 или не ранее чем через год с начала внедрения вышеописанных методик.

Выводы. Оптимизация тестирования – это многогранный процесс, который требует комплексного подхода и внимательного учета всех аспектов работы организации. Прежде всего, важно понимать, что тестирование — это не просто проверка продукта на наличие ошибок, а полноценный процесс управления качеством, в котором интеграция различных методологий и подходов играет ключевую роль. В условиях ООО «Энерговектор» оптимизация процессов тестирования особенно актуальна, поскольку данная организация, как и многие другие, сталкивается с вызовами на каждом этапе разработки и проверки программного обеспечения.

Целесообразность использования различных моделей, таких как CMMI, TMAP, TPI Next и других, обусловлена потребностью в стандартизации процессов, повышении уровня зрелости команды и увеличении эффективности тестирования.

Общие модели управления процессами, такие как CMMI, подходят для более длительных и комплексных проектов, требующих вовлечения крупных команд и руководства, тогда как узкоспециализированные модели (например, TMMi) сосредоточены на совершенствовании отдельных аспектов тестирования.

Помимо использования моделей, аналитические подходы, такие как Диаграмма Исикавы, метод «5 Почему» и GQM, играют не менее важную роль. В частности, аналитические подходы незаменимы для установления причинно-следственных связей, что позволяет оперативно реагировать на ошибки и предотвращать их повторение в будущем.

Ниже более подробно остановимся на метриках, которые используются в ООО Энерговектор, позволяя повысить эффективность и производительность тестирования ПО.

Метрики, такие как Defect Detection Percentage и Cost of Quality Ratio, предоставляют руководству и команде тестировщиков объективные данные, на основе которых можно принимать обоснованные решения по оптимизации.

Гибридные модели оптимизации тестирования, сочетающие в себе различные подходы и методологии, обеспечивают наиболее гибкий и адаптивный процесс внедрения изменений.

В рамках ООО «Энерговектор» такой гибридный подход позволит учитывать уникальные особенности организации и одновременно воспользоваться преимуществами различных методик. Например, сочетание моделей управления процессами с аналитическими подходами создаст условия для более системного и всеобъемлющего улучшения качества продукта.

Выводы по второй главе

Таким образом, внедрение комплексной стратегии оптимизации тестирования позволит ООО «Энерговектор» не только повысить качество разрабатываемого программного обеспечения, но и существенно улучшить эффективность работы всей команды.

Глава 3 Разработка стратегий и методов тестирования ПО

3.1 Основные бизнес-цели и методики организации процесса тестирования ПО в ООО Энерговектор

Некоторые ключевые бизнес-цели тестирования программного обеспечения включают в себя:

- оценка рабочих продуктов. Анализируются требования, пользовательские истории, проектная документация и исходный код;
- проверка соответствия требованиям. Необходимо убедиться, что все зафиксированные требования были выполнены;
- оценка готовности объекта тестирования. Проверяется, завершено ли разработанное решение и соответствует ли оно ожиданиям пользователей и заинтересованных сторон [50], [56].

Одной из ключевых целей процесса тестирования является формирование уверенности в уровне качества тестируемого объекта, достигаемого путём систематического анализа его характеристик и соответствия установленным требованиям.

Не менее важным результатом тестирования становится предотвращение дефектов, достигаемое за счёт выявления потенциальных проблемных зон ещё на ранних этапах разработки. При этом гарантируется, что допущенные ошибки, подвергшиеся корректировке, не повторятся ни в текущем проекте, ни на последующих стадиях разработки аналогичных продуктов, что существенно повышает общую надёжность программного обеспечения.

Тестирование также играет важнейшую роль в предоставлении заинтересованным сторонам достаточной информации, позволяющей принимать обоснованные решения о качестве продукта. Разработанные отчёты, содержащие данные о результатах тестирования, уровне выявленных

дефектов и степени их критичности, служат основой для оценки готовности системы к выпуску или дальнейшим доработкам.

Кроме того, данный процесс направлен на снижение рисков, связанных с ненадлежащим качеством программного обеспечения, которые могут привести к финансовым потерям, утрате репутации компании или возникновению угроз безопасности данных. Выявляя и устраняя дефекты на этапах тестирования, разработчики предотвращают возможные негативные последствия, которые могли бы возникнуть в процессе эксплуатации.

Наконец, одной из приоритетных задач тестирования является проверка соответствия тестируемого объекта договорным, правовым и нормативным требованиям. Данный этап, включающий анализ соблюдения стандартов, регламентов и лицензионных соглашений, обеспечивает юридическую и техническую надёжность продукта, гарантируя его соответствие как внутренним ожиданиям заказчика, так и внешним регламентирующим документам.

Рассмотрим методы оценки объема проекта тестирования в ООО Энерговектор. Для оценки объема тестирования проекта в ООО Энерговектор применяются следующие методы.

Экспертная оценка. Опытные члены команды определяют объем тестирования, исходя из своего опыта и знаний о проекте. Оценка по точкам (Story Points). Каждая задача или итерация получает оценку в виде точек, отражающих сложность и объем работ.

Метод Planning Poker. Участники команды оценивают объем работы, используя специальные карточки. Обсуждение оценок продолжается до достижения согласия.

Метод 3T (Test Types, Test Levels, Test Techniques). Задачи разделяются на три категории: типы тестов, уровни тестирования и техники тестирования. Каждая категория оценивается отдельно, после чего результаты суммируются для получения общего объема тестирования.

Оценка на основе тестовых сценариев. Оценка основывается на количестве тестовых сценариев или кейсов, которые нужно разработать и выполнить. Данный метод включает в себя анализ требований и определение необходимых сценариев для полного охвата функциональности.

Комбинированный метод. Для получения более точной оценки можно сочетать несколько из вышеупомянутых методов. Например, начать с экспертной оценки, а затем использовать оценку по точкам для уточнения [62], [67].

Формула для определения точного объема тестирования отсутствует. В большинстве случаев уровень тестирования можно установить только на основе консенсуса между руководителем проекта, заказчиком, разработчиками, техническими специалистами и тестировщиками.

Карта пути клиента ООО Энерговектор

Карта пути клиента (CJM) служит для визуализации процесса, в ходе которого клиент, начиная с момента осознания потребности в продукте и заканчивая его покупкой, а порой и последующим взаимодействием, проходит через ряд этапов. Переходя от одного этапа к другому, клиент взаимодействует с продуктом и компанией, формируя своё мнение и принимая решения на основе накопленного опыта.

Помимо CJM, применяются и другие карты, такие как UJM (user journey map) и LXM (life experience map), каждая из которых играет свою роль в процессе анализа. UJM, описывая онлайн-взаимодействия пользователя с сайтом или приложением, фокусируется на его действиях и взаимодействиях в цифровой среде. LXM, в свою очередь, предоставляет информацию о жизни потенциальных клиентов, позволяя лучше понять их интересы, проблемы и потребности, что необходимо для улучшения продукта и оптимизации взаимодействий на каждом этапе, включая тех, кто ещё не стал клиентом. Эти инструменты необходимы для того, чтобы максимально точно определить, какие изменения будут наиболее эффективны для удовлетворения и привлечения клиентов.

В настоящее время карта пути клиента находится на этапе разработки в руководстве ООО "Энерговектор". Ниже представлены шаблоны, рекомендуемые для заполнения карты CJM (рис. 4).

Рисунок 4 - Шаблоны диаграмм CJM для ООО Энерговектор

Карта эмпатии ООО Энерговектор

Картирование эмпатии, служащее важным инструментом в инновационной сфере IT, обеспечивает глубокое понимание человеческих потребностей и препятствий. Действуя как компас для дизайнеров, эта методика позволяет систематически анализировать мысли и поведение пользователей, превышая границы обычных статистических данных. Визуальное представление карты эмпатии помогает командам, в том числе в ООО "Энерговектор", глубже сочувствовать целевой аудитории, создавая продукты, которые находят отклик у пользователей. Такой подход способствует формированию всестороннего взгляда на опыт пользователей,

позволяя разработчикам проектировать уникальные пользовательские впечатления.

Как правило, карта эмпатии состоит из четырех секторов, каждый из которых сосредоточен на различных аспектах опыта пользователя (рис. 5).

Рисунок 5 - Карта эмпатии

Давайте рассмотрим создание карты эмпатии на примере разработки приложения для предварительного проектирования газификации. Представим команду UX-дизайнеров, работающих над этим приложением, которые стремятся понять мотивы и опыт своей целевой аудитории.

Карта эмпатии может быть заполнена следующей информацией.

Говорит: (Цитаты пользователей из интервью)

- "Мне нужно приложение для газификации, чтобы рассчитать мои приблизительные расходы по подключению газа в моем доме за городом."

- "Хочу подать заявку на газификацию онлайн и максимально простым способом."

Думает: (Стремления и опасения пользователя)

- Стремление: "Хочу скорее подключить газ. Скоро зима. Отапливаться газом выгодно."

- Опасение: "Я беспокоюсь о том, что расходы на газификацию на моем участке будут слишком высокими. Хочу выяснить, есть ли в нашем районе проложенные газовые трубы. Тогда газафикация дома будет дешевле".

Чувства: (Эмоции пользователя)

- Испытывает удовлетворение от возможности подключить газ недорого, просто и быстро.

- Испытывает разочарование из-за предыдущих неудач с поисками планов газификации своего района.

Делает: (Действия пользователя)

- Следит за новостями газификации своего района в социальных сетях.

Эта карта эмпатии (рис. 6) поможет команде разработчиков и тестировщиков глубже понять потребности своих пользователей и разработать приложение предварительного проектирования газификации загородных домов, которое будет эффективно и удобно для целевой аудитории.

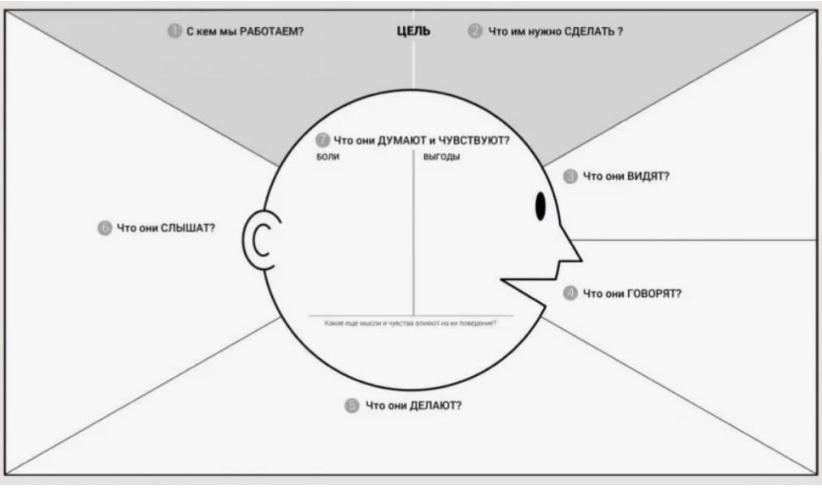

Рисунок 6 - Карта эмпатии ООО Энерговектор

Рассмотрим матрицу ролей ООО Энерговектор.

Матрица ролей в тестировании служит для четкого определения ролей и обязанностей всех участников процесса, что помогает избежать неопределенности при выполнении задач и действий.

Одним из используемых вариантов матрицы ролей является матрица RACI. Эта таблица представляется в следующем формате: по вертикали перечисляются задачи проекта, а по горизонтали — исполнители. На пересечении задач и исполнителей помещаются буквы, обозначающие роли в проекте и уровень ответственности.

Некоторые ключевые роли в тестировании и их описание представлены на схеме (рис. 7).

Рисунок 7 - Ключевые роли в тестировании и их описание

Матрица ролей в ООО Энерговектор разрабатывается и согласовывается на начальном этапе проекта, что позволяет избежать

ситуации, когда исполнители перекладывают ответственность друг на друга (рис. 8-10).

	Управление продуктом	Управление программой	Разработка	Тестирование	Удовлетворение потребителя	Управление выпуском
Управление продуктом		-	-	+	+	±
Управление программой	-		-	±	±	+
Разработка	-	-		-	-	-
Тестирование	+	±	-		+	+
Удовлетворение потребителя	+	±	-	+		±
Управление выпуском	±	+	-	+	±	

\+ Возможно ± Нежелательно - Нельзя

Рисунок 8 - Матрица совместимости ролей (MSF) ООО Энерговектор

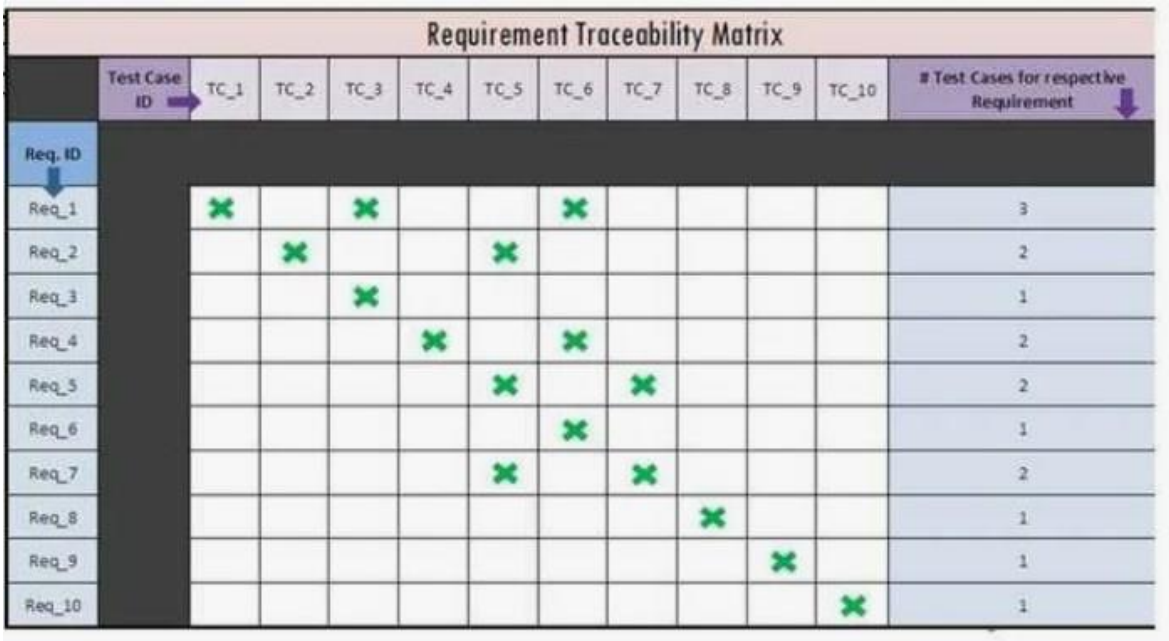

Рисунок 9 - Матрица соответствия ролей тестовых сценариев (test cases) и функциональных требований (requirements) ООО Энерговектор

Матрица RACI	Сотрудник				
Функция	**Святослав**	**Анастасия**	**Василий**	**Татьяна**	**Вангьял**
Создание плана	A	R	I	I	I
Сбор требований	I	A	R	C	C
Разработка плана изменений	I	A	R	R	C
Разработать план тестирования	A	C	I	I	R

R - Responsible; A - Accountable; C - Consulted; I - Informed

Рисунок 10 - Матрица ролей RACI ООО Энерговектор

Рассмотрим дерево целей ООО Энерговектор (рис. 11).

Рисунок 11 - Дерево целей общей деятельности ООО Энерговектор

Дерево целей представляет собой метод планирования, позволяющий разбить крупную цель на набор небольших взаимосвязанных и легко понимаемых шагов.

В контексте тестирования данная древовидная структура дает возможность создавать чек-листы и тест-кейсы в зависимости от конкретных потребностей проекта (рис. 12).

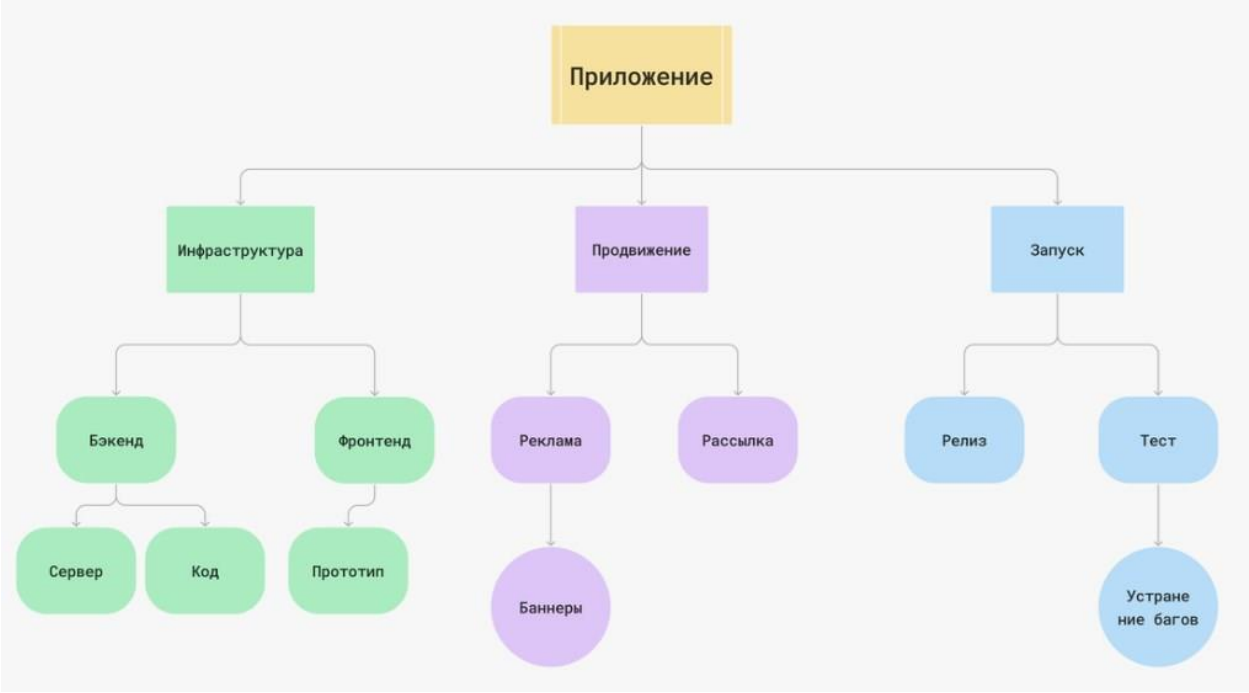

Рисунок 12 - Дерево целей разработки приложений и тестирования ООО Энерговектор

Вот несколько подходов к её применению:

Первый подход, основанный на структуре экранов приложения, предполагает, что на первом уровне дерева располагаются тестовые кейсы, соответствующие основным разделам интерфейса приложения. Разделив приложение на отдельные экраны, к каждому из них добавляют перечень состояний, в которых тот может находиться, что облегчает проведение тестирования. На более глубоких уровнях дерева формируется чек-лист, представленный в табличном формате, где подробно описываются элементы дизайна и способы взаимодействия с ними, что позволяет эффективно проверять корректность функционирования интерфейса.

Второй подход, сосредотачивающий внимание на объектах и действиях, отклоняется от навигационной структуры и концентрируется на документации API и бизнес-логике приложения. В данном случае негативные и позитивные сценарии тестирования выстраиваются по отдельным ветвям дерева, что способствует более глубокому анализу взаимодействий между различными объектами.

Третий метод, основанный на use-cases, предполагает декомпозицию функционала приложения в зависимости от пользовательских сценариев. На первом этапе анализируются цели, которые могут преследоваться различными типами пользователей. Затем, на основе определённых ролей и задач, формируются ветви дерева, охватывающие ключевые моменты взаимодействия с приложением, что позволяет сосредоточиться на проверке наиболее значимых и частых сценариев использования.

Выбор стратегии зависит от потребностей проекта и приоритетов заказчика.

3.2 Бизнес-процессы и стратегии автоматизации тестирования в ООО Энерговектор

В рамках процесса автоматизации тестирования выделяют следующие основные этапы (рис. 13).

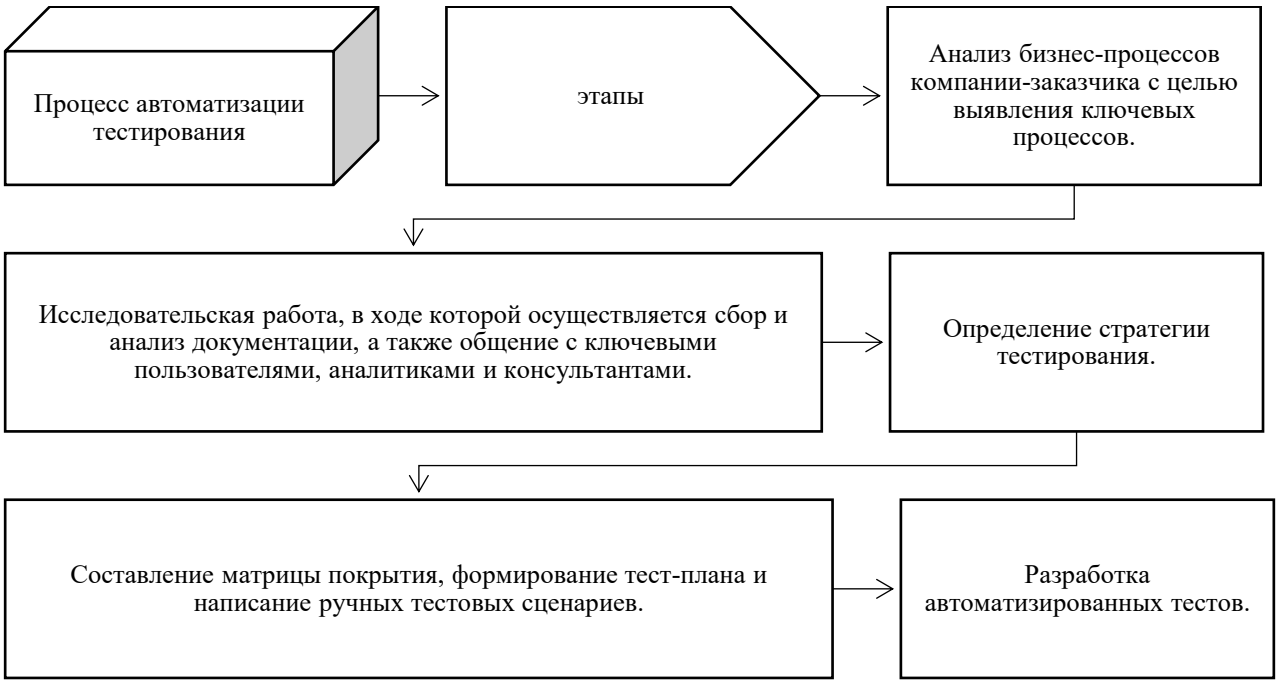

Рисунок 13 - Этапы процесса автоматизации тестирования

В системе SAP применяются инструменты, такие как SAP CBTA, SAP ECATT, Unified Functional Testing (UFT), IBM Rational Functional Tester, SmartBear TestComplete и другие.

Ниже представлены информационно-ресурсная модель IDEF 0 бизнес-процессов в компании ООО «Энерговектор» (рис. 14), модель построения процессов тестирования в ООО "Энерговектор" Как Есть (рис. 15) и модель построения процессов тестирования в ООО "Энерговектор" Как Должно быть (рис. 16).

.

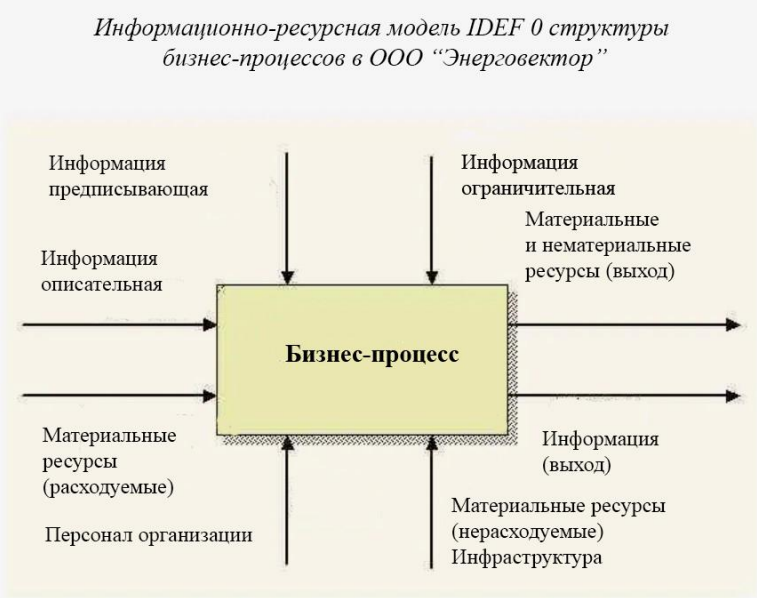

Рисунок 14 - Модель IDEF 0 бизнес-процессов автоматизации тестирования в ООО "Энерговектор"

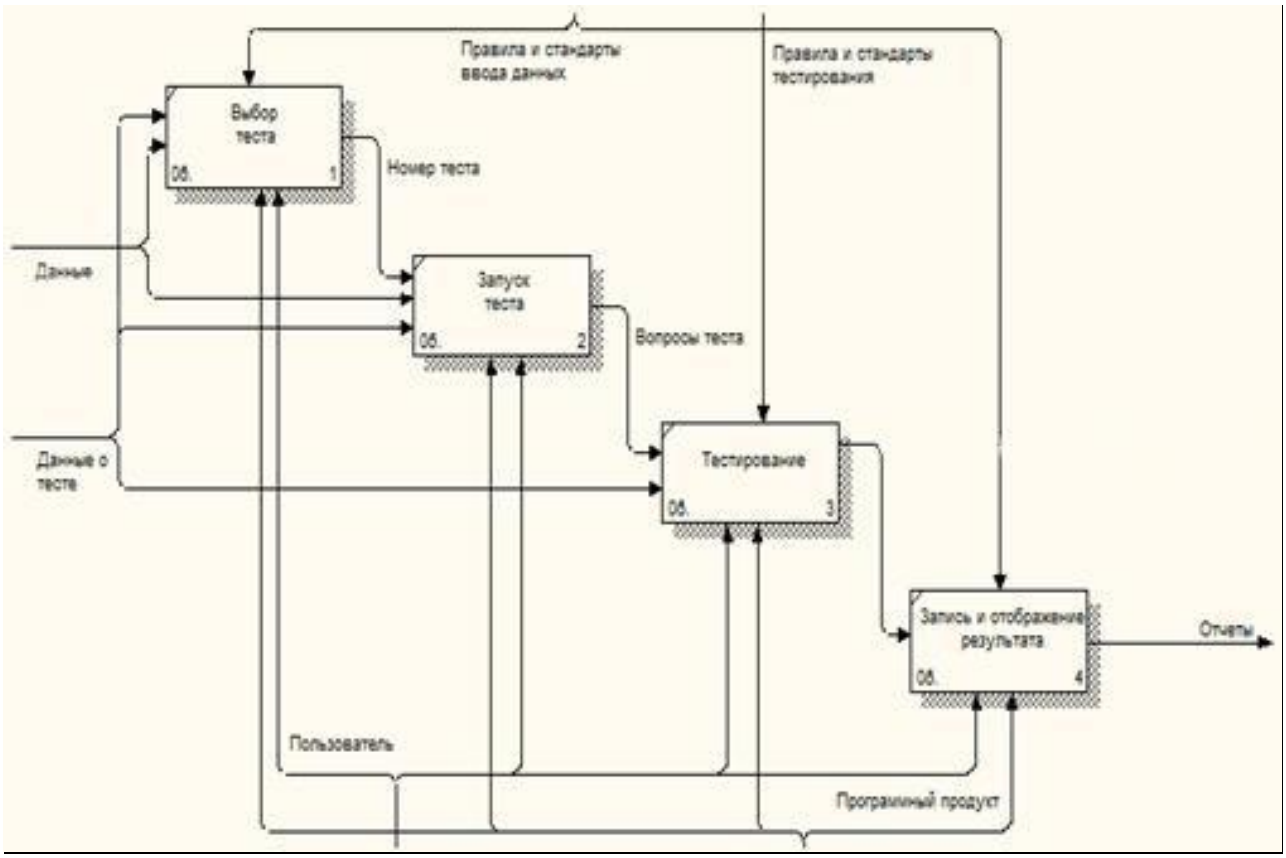

Рисунок 15 - Модель построения процессов тестирования в ООО "Энерговектор" (Как Есть)

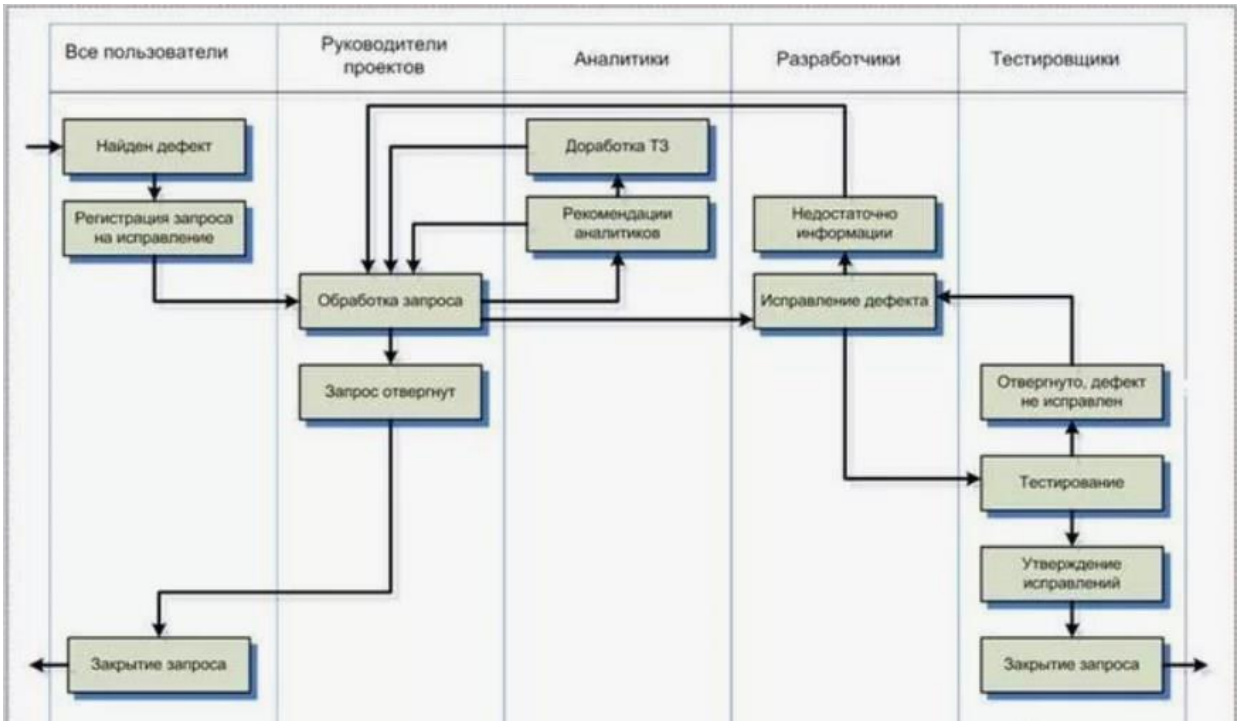

Рисунок 16 - Модель построения процессов тестирования в ООО "Энерговектор" (Как Должно быть)

Декомпозиция процессов тестирования ПО.

Функциональная декомпозиция, используемая в процессах тестирования программного обеспечения, представляет собой методологический подход, в рамках которого комплексные операции разбиваются на более мелкие и управляемые элементы. Стандарт предусматривает создание контекстной диаграммы первого уровня, на которой центральное место занимает задача тестирования ПО. Эта диаграмма демонстрирует не только основную цель процесса, но и визуализирует потоки входных и выходных данных, механизмы контроля и управления этими данными, а также различные ограничения, влияющие на процесс.

Такая структурированная подготовка и анализ процесса тестирования позволяют точнее определить задачи и ответственности, улучшить координацию действий команды и повысить эффективность обнаружения и исправления ошибок в программном продукте. Это, в свою очередь,

способствует повышению качества конечного продукта и удовлетворенности пользователя. Ниже представлены контекстная диаграмма верхнего уровня процесса тестирования ПО (рис. 17) и диаграмма декомпозиции процессов тестирования ПО в ООО "Энерговектор" (рис. 18).

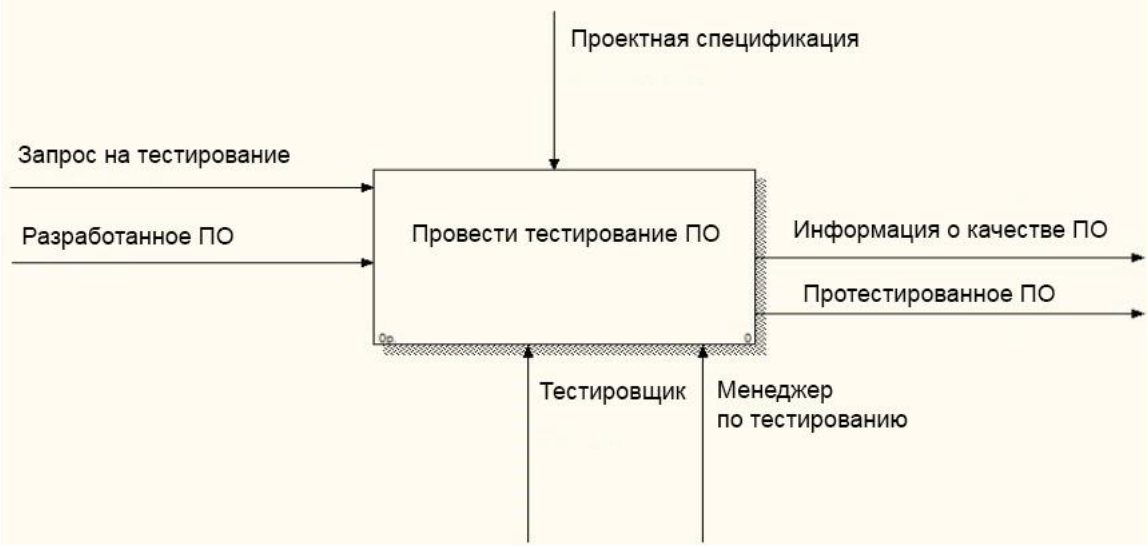

Рисунок 17 - Контекстная диаграмма верхнего уровня процесса тестирования ПО

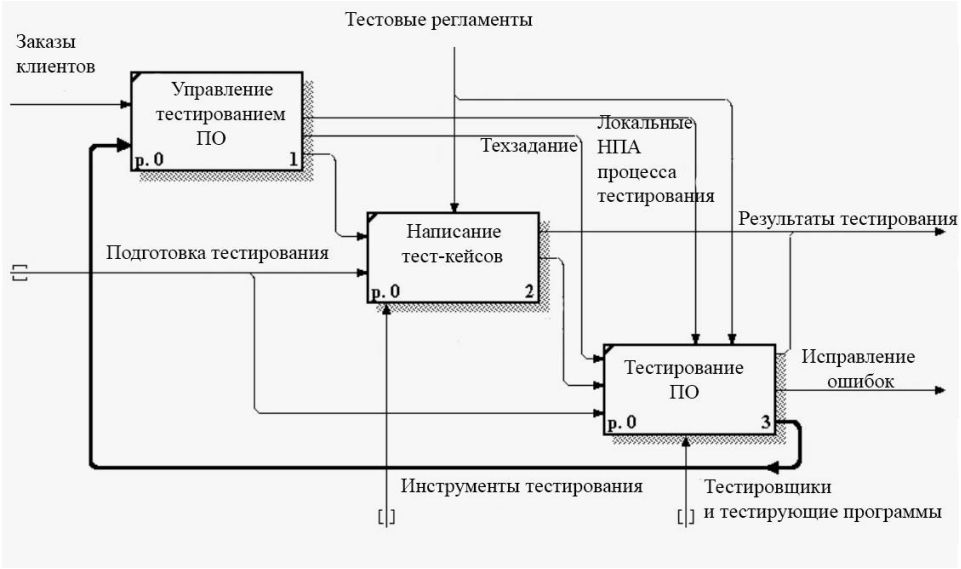

Рисунок 18 - Диаграмма декомпозиции процессов тестирования ПО в ООО "Энерговектор"

Тестирование ПО начинается с составления запроса на тестирование, разработанного с учетом нового и старого ПО и их совместимости. Далее составляется план тестирования, который следует согласовать с менеджером разработки. Следующим этапом становится подбор команды, формируемой с учетом требуемых компетенций для оценки различных аспектов программного продукта. В ходе тестов собираются количественные метрики, позволяющие объективно оценить качество ПО. По завершении цикла тестирования выдаётся заключение о качестве продукта, что играет важную роль в предотвращении ошибок, потенциально влияющих на систему. Подбор квалифицированной команды тестировщиков является ключевым для выявления несоответствий и гарантирует успешное тестирование, что снижает затраты на поддержку и исправление ошибок в дальнейшем [64].

Процесс анализа полученных количественных метрик, таких как время отклика системы, уровень производительности и стабильности, даёт возможность объективно оценить, насколько продукт соответствует заявленным требованиям и ожиданиям пользователей. Таким образом, итоговый документ с заключением о качестве ПО, формируемый на основе проведенных тестов, становится основой для принятия решений о выпуске продукта, определяя готовность системы к эксплуатации.

Не менее важным аспектом является правильный подбор персонала на проект, что напрямую влияет на успешное выполнение задач в рамках тестирования. Критерии подбора специалистов, соответствующих требованиям проекта, позволяют создать команду, которая будет эффективно решать поставленные задачи, снижая вероятность появления ошибок, связанных с человеческим фактором. Таким образом, каждый из перечисленных этапов, будучи взаимосвязанным с другими, является критически важным для обеспечения высокого качества программного продукта.

На рисунке ниже представлена декомпозиция контекстной диаграммы (рис. 19).

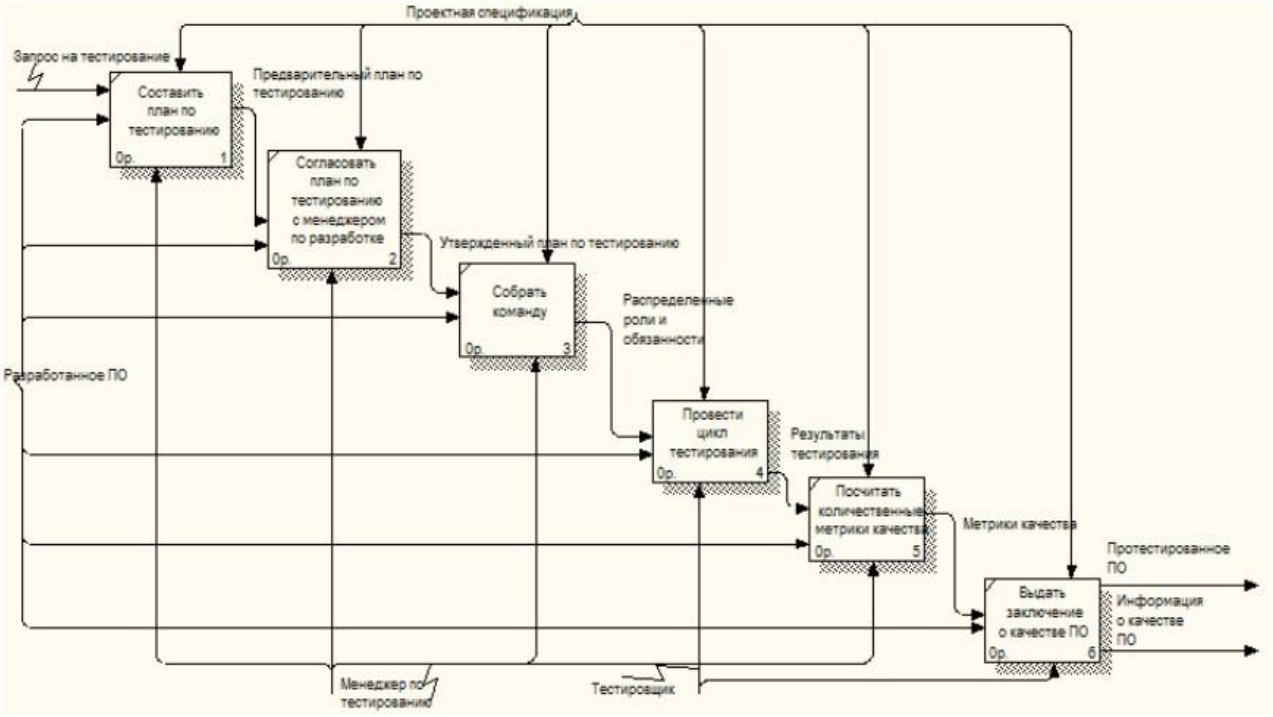

Рисунок 19 - Декомпозиция первого уровня процесса тестирования ПО

На рисунке 20 представлена диаграмма декомпозиции «Выполнить цикл тестирования». На диаграмме показаны различные этапы тестирования: полный тест приложения, кроссбраузерное тестирование и т.д.

После этого тестировщик выполняет валидацию обнаруженных дефектов и формирует список тех, которые прошли валидацию. Завершающим этапом является регрессионное тестирование.

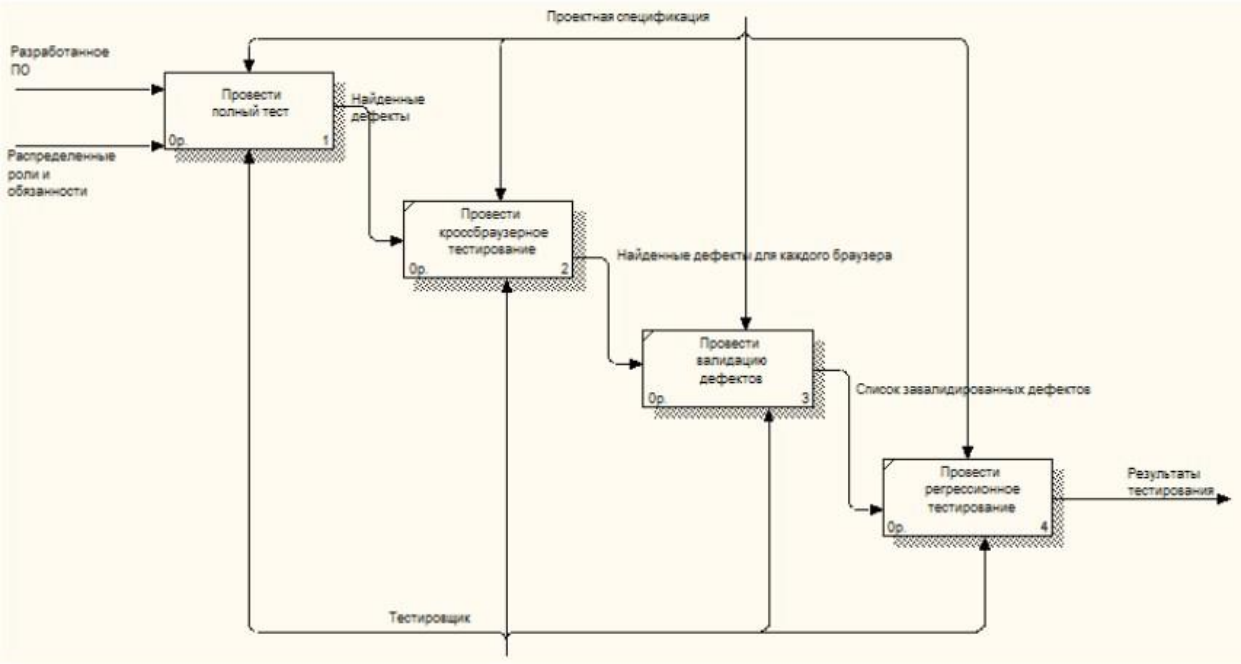

Рисунок 20 - Диаграмма декомпозиции «Выполнить цикл тестирования»

Тестирование приложения, выполняя центральную роль в процессе обеспечения качества программного обеспечения, способствует выявлению множества проблем, которые могут негативно сказаться на функциональности и стабильности работы продукта. Этот процесс, заключающийся в систематическом анализе работы ПО и устранении обнаруженных ошибок, позволяет обеспечить соответствие приложения заявленным требованиям.

Правильно организованное тестирование, включающее проверку различных аспектов работы приложения, таких как функциональность, производительность и безопасность, даёт возможность не только обнаружить существующие дефекты, но и предотвратить их появление на последующих этапах разработки. Создание качественной функциональности, являясь неотъемлемым элементом разработки, становится основой для достижения высокой степени удовлетворения потребностей пользователей.

Результатом комплексного анализа, проводимого в рамках тестирования, становятся тщательно спроектированные модели, использующиеся для разработки методов оценки качества приложения.

3.3 Разработка метода компонентного тестирования в ООО Энерговектор

Компонентное тестирование представляет собой один из видов тестирования модели автоматизации тестирования ООО Энерговектор, изображенной на рис. 21.

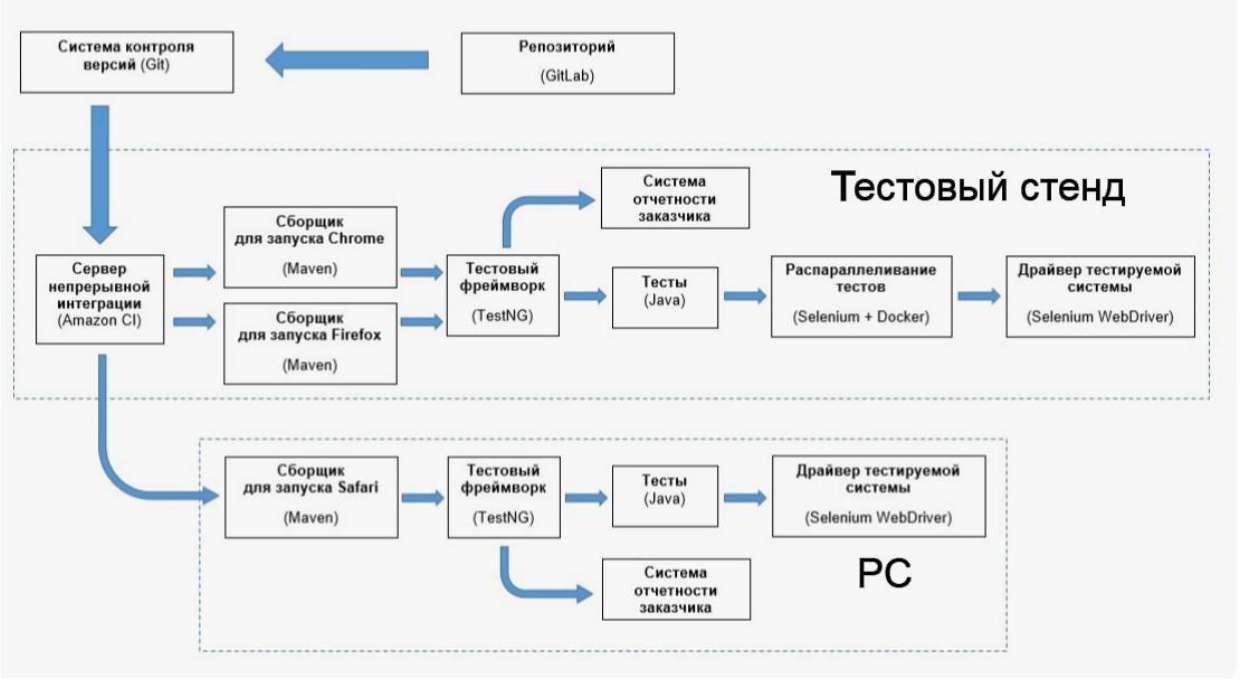

Рисунок 21 - Модель процессов автоматизированного тестирования ООО Энерговектор

Этот тип тестирования программного обеспечения, в рамках которого проверяется каждый отдельный компонент в изоляции от других. Основная цель компонентного тестирования заключается в том, чтобы подтвердить правильность функционирования каждого компонента и его соответствие

заданным требованиям. Пример компонентного тестирования с учетом производительности представлен на рис. 22.

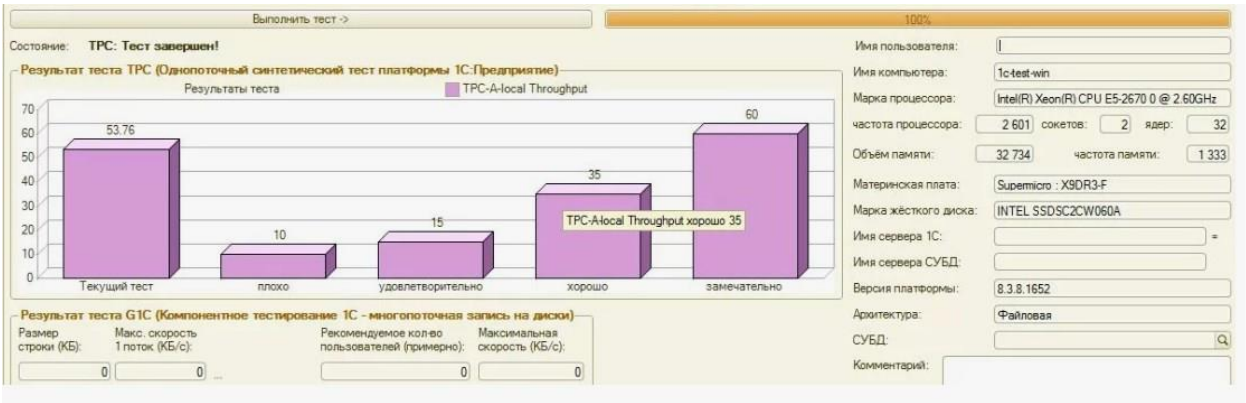

Рисунок 22 - Компонентное тестирование с учетом производительности.

Метод компонентного тестирования широко используется специалистами QA в процессе тестирования по методике «Чёрного ящика».

Компонентное тестирование можно классифицировать в зависимости от глубины уровней тестирования следующим образом:

- Тестирование компонентов в малом масштабе (CTIS — Component Testing In Small) — оно может проводиться как с изоляцией, так и без изоляции прочих компонентов в тестируемом программном обеспечении или приложении.

- Тестирование компонентов в большом масштабе (CTIL — Component Testing In Large) — это тестирование компонентов, выполняемое без изоляции других компонентов в тестируемом программном обеспечении или приложении.

Мутационное тестирование и компонентное тестирование часто рассматриваются в контексте их способности обнаруживать ошибки в программном обеспечении, однако каждый из этих методов имеет свои уникальные подходы и области применения. Мутационное тестирование основывается на принципе внесения небольших изменений (мутаций) в исходный код программы, целью которых является имитация возможных

ошибок, допускаемых программистами. Применение так называемых мутирующих операторов к различным элементам программы приводит к созданию мутантов — изменённых версий исходной программы, которые должны быть обнаружены с помощью набора тестов. Эффективность мутационного тестирования оценивается через способность тестов выявить и "убить" мутантов, то есть найти различия между поведением оригинальной и мутированной программ.

Для того чтобы тестирование можно было считать успешным, должны быть выполнены три условия, определяемые RIP-моделью: достижение мутирующего оператора, заражение состояния программы и распространение этого изменения на вывод программы, который затем проверяется тестом. Таким образом, RIP-модель является основой сильного мутационного тестирования, которое требует выполнения всех этих условий и обеспечивает максимальную гарантию обнаружения ошибок.

Однако одним из главных недостатков метода сильного мутационного тестирования является высокая вычислительная сложность, обусловленная необходимостью генерации огромного количества мутантов. Для небольших программ это число может достигать десятков тысяч, что приводит к значительным затратам времени и ресурсов на их тестирование. В связи с этим разработчики вынуждены искать компромиссные решения, одним из которых является компонентное тестирование, представляющее собой приближенную интерпретацию метода сильной мутации.

Компонентное тестирование фокусируется на трансформациях отдельных компонентов программы, таких как арифметические выражения, ссылки на переменные и логические отношения. В этом методе набор тестов направлен на определённые вычислительные структуры программы, что позволяет сократить количество трансформируемых элементов по сравнению с полным мутационным тестированием. По своей сути компонентное тестирование является упрощённой версией сильного мутационного

тестирования, но при этом сохраняет его основные принципы и гарантирует достаточно высокую эффективность при обнаружении ошибок.

Можно отметить, что структурное тестирование, в частности ветвевое тестирование, является частным случаем компонентного подхода.

Избирая предикаты в качестве компонентов для изменений, мы искусственно повышаем эффективность компонентного тестирования, которое в условиях ветвления и генерации циклов приближается по своим техническим параметрам к методике путевого тестирования.

Таким образом, методы компонентного тестирования могут быть частично упорядочены в зависимости от их способности выявлять ошибки, причём сильная мутация занимает высшую позицию в этом упорядочении, выступая как теоретически максимальная граница для других методов.

Булевские выражения. Булевое выражение — это арифметическое выражение, использующее только два значения: 1 и 0, или True и False. Этот подход был предложен математиком Джорджем Булем и применяется для объединения истинных и ложных условий (представляемых как 1 и 0) с помощью логических операторов and, or и not.

Некоторые ключевые логические операции представлены на схеме (рис. 23).

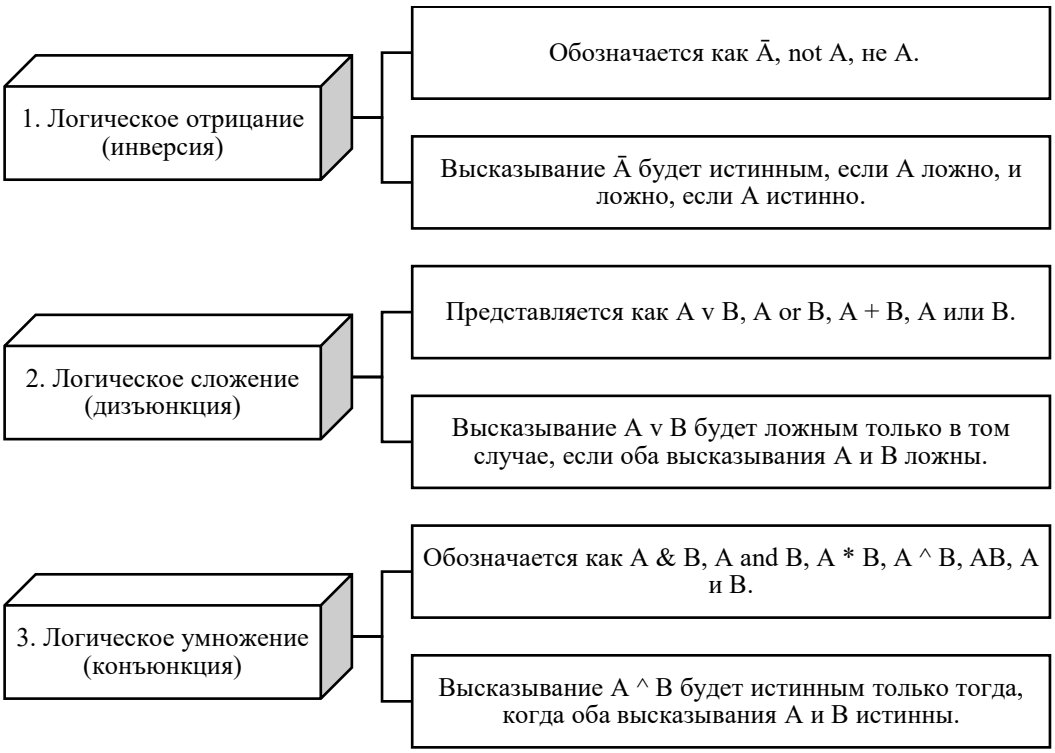

Рисунок 23 - Ключевые логические операции

3.4 Программные конструкции и их тестирование

Методы тестирования простых программных компонентов, описанные выше, могут быть эффективно применены для выявления ошибок в более сложных конструкциях, таких как операторы присваивания, условные выражения и циклы. Оператор присваивания, являясь ключевой конструкцией, осуществляет передачу значения переменной или элементу массива посредством функций, связанных с соответствующими элементами. При этом переменные, массивы и индексные выражения, указанные в правой части оператора, взаимодействуют с функциями, обеспечивающими доступ к данным.

Условный оператор. В процессе компонентного тестирования условия проверяются как с истинными/ ложными значениями. Данный метод относится к структурному тестированию и предполагает анализ всех независимо выполняемых ветвей компонента или программы. Оператор цикла. Оператор цикла является важной частью программного кода. Корректная работа этого оператора напрямую влияет на выполнение всего алгоритма. На этапе модульного тестирования часто выявляются дефекты, возникающие из-за ошибок циклах, что может быть связано с неправильным пониманием условий их завершения, использованием некорректных счетчиков или нарушением работы с локальными переменными. Такие ошибки, проявляясь в процессе тестирования, могут приводить к бесконечным или преждевременно завершающимся циклам, а также к ошибкам доступа к памяти или некорректному использованию ресурсов. Циклическое тестирование делится на три категории в зависимости от типов циклов (рис. 24).

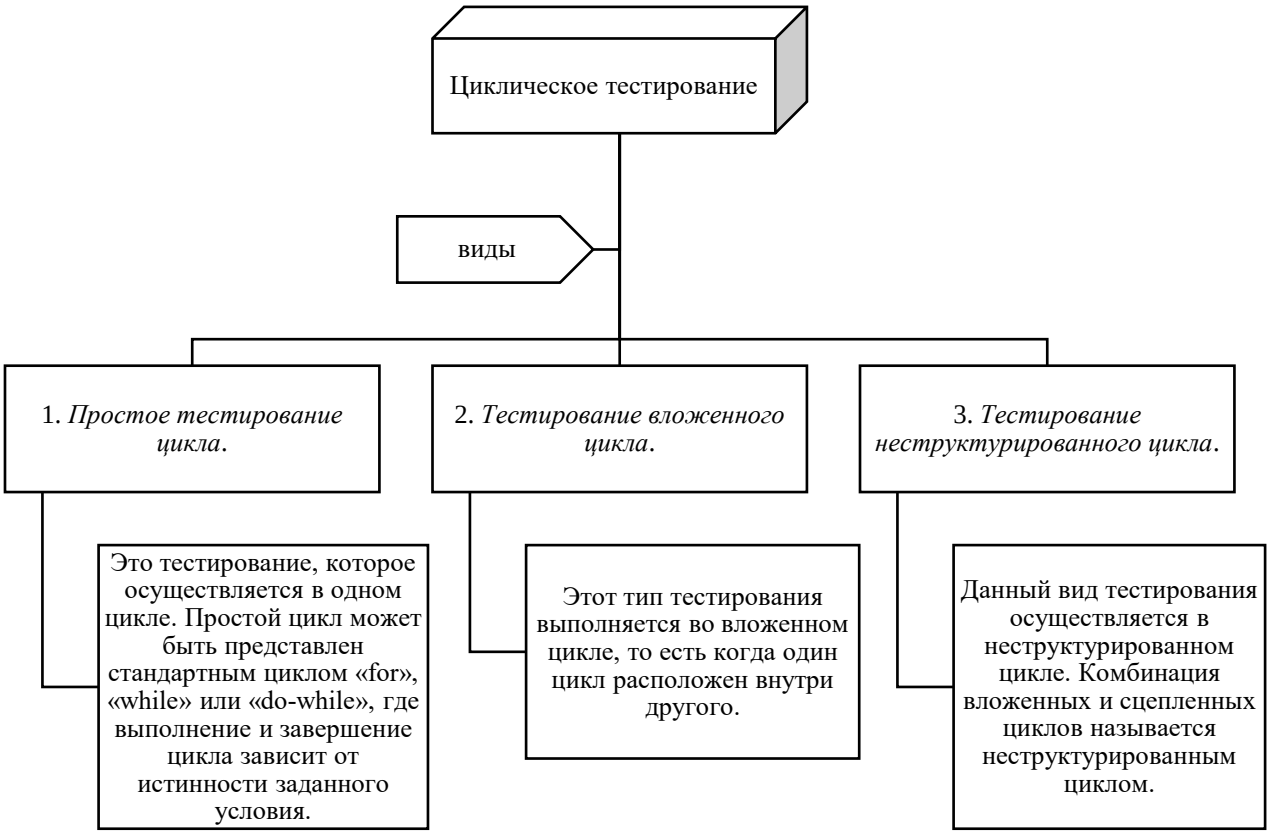

Рисунок 24 - Виды циклического тестирования

При этом при тестировании функций отношений, применяемых в циклических операторах, необходимо предусмотреть тесты, способные выявить ошибки, связанные с неверным определением момента входа в цикл. Так, тестовое множество должно содержать сценарии, в которых ожидается начало цикла, но программа его не запускает, либо, напротив, цикл начинается, когда это не предполагается. Для функций, отвечающих за завершение цикла, тесты должны быть направлены на выявление некорректного выхода из цикла, например, в случаях преждевременного завершения или, наоборот, продолжения цикла при необходимости его завершения.

Вместе с тем, следует учитывать, что создание полного тестового множества может быть недоступно в силу сложности системы или других объективных ограничений.

3.5 Эффективность и полнота методов компонентного тестирования

Эффективность тестовых множеств является центральной концепцией в процессе тестирования программного обеспечения, нацеленной на выявление потенциальных ошибок в системе. В работах Гуденафа и Герхарта она была определена через два взаимосвязанных аспекта: надежность и значимость. Так, тестовое множество считается надежным, если использование данного теста гарантированно приводит к обнаружению ошибок, если таковые присутствуют в программе. Это подразумевает, что каждый тест должен быть способен идентифицировать специфичные неисправности в определенной части кода.

Продолжая развитие этой идеи, Вейкар и Остранд внесли дополнительные уточнения в концепцию, введя понятие обнаруживающей подобласти — части тестового множества, которая особенно эффективна в выявлении определенных видов ошибок. Такое дополнение позволило более детально анализировать и оценивать тесты с точки зрения их способности обнаруживать конкретные недостатки в коде, что, в свою очередь, способствует повышению качества программного продукта.

Таким образом, определяя тестовое множество через призму его надежности и значимости, исследователи смогли предложить методы и инструменты для более целенаправленного и эффективного тестирования, обеспечивающего высокий уровень уверенности в отсутствии критических ошибок в разрабатываемом программном обеспечении.

Полнота тестовых множеств. Концепции полноты тестовых множеств формируют совокупность систематических методов тестирования, использование которых оставляется на усмотрение тестировщика.

Ветвевое тестирование, предоставляя обширные возможности для анализа, позволяет эффективно сравнивать его с компонентным тестированием, демонстрируя, в частности, почему последнее может

рассматриваться как уточнение первого. Важно отметить, что успешное выявление ошибок в программном коде требует не просто прохождения по определенной ветке кода в рамках теста, но и применения тестовых данных, соответствующих различным видам ошибок операторов и кода. В ветвевом тестировании системы множеств Sf сводятся к обнаружению грубых дефектов, выявляемых на стадии активации операторов. В свою очередь множества компонентного тестирования отличаются большей точностью и охватом значительного разнообразия мелких дефектов, распознаваемых в процессе скрупулезного анализа ветвевой функциональности и конкретных значений тестирования. Применяемые в методе компонентного тестирования множества P имеют более значительный размах, выходя за рамки булевских функций, что в свою очередь расширяет возможности тестирования, улучшая обнаружение ошибок на различных уровнях исполнения кода. Это позволяет более тщательно анализировать поведение компонентов программы и эффективно устранять потенциальные уязвимости.

Итак, подводя итог исследования, сформулируем основные выводы.

В рамках исследования были выделены и проанализированы основные компоненты, подверженные возникновению дефектов; были разработаны методики оценки их адекватности, предложены усовершенствованные алгоритмы тестирования для их оперативного обнаружения.

Введенные функции для анализа простейших элементов программ позволили разработать механизм оценки более сложных структур, таких как операторы присваивания и циклы. Также были исследованы вопросы эффективности и полноты применяемых методов тестирования в контексте указанных ошибок.

Научная новизна исследования. В данном исследовании рассматриваются различные подходы к тестированию программного обеспечения: в т.ч. тестирование, независимое от спецификаций, основанное на спецификациях и аппаратных средств. На основе этих методов была предпринята попытка создать методику компонентного тестирования для

информационных систем. В рамках научно-исследовательской работы было изложено определение комплекса действий данного метода, выделены критерии для компонентного тестирования, а также сформулирована спецификация компонентного тестирования.

Научная новизна данной научно-исследовательской работы заключается в следующих моментах:

- разработана улучшенная модель программного интерфейса, представляющая структуру и функции интерфейса с опорой на расширенный метод контрактных спецификаций (на базе единого типа данных);

- разработан метод автоматизации тестирования ПО с акцентом на автоматизацию процесса выстраивания модели API-интерфейса, расширение и улучшение функционала в рамках спецификаций и с учетом структуры поэтапной генерации тестового кода;

- создан алгоритм для визуализации усовершенствованной модели, направленной на анализ спецификаций интерфейса;

- предложен алгоритм обхода, отвечающий за автоматическое создание тестов в виде цепочки вызовов методов и условий проверки ПО и его функций. Такой предложенный нами алгоритм направлен на повышение качества управления тестовым покрытием разработанной модели;

- предложен и успешно протестирован алгоритм оптимизации тестов, делающий упор на непрерывный анализ тестового набора.

В проведенном нами исследовании рассматриваются различные подходы к тестированию ПО, включая тестирование, независимое от спецификаций, тестирование, основанное на спецификациях, и тестирование аппаратных средств. На основе этих методов была предпринята попытка создать методику компонентного тестирования для информационных систем. В данной работе было предложено уточненное определение комплекса действий данного метода, выделены дополнительные критерии компонентного тестирования, обозначены уточненные спецификации компонентного тестирования.

Помимо этого, в данной ВКР было сформулировано уточненное определение совокупности действий метода, описаны актуальные в работе ООО Энерговектор программные составляющие и их мутации (переменные и их присваивание, арифметические выражения, логические отношения, булевские выражения и пр.). В работе проанализированы программные конструкции и процесс их тестирования в ООО Энерговектор (операторы циклов, условные и присваивания); уточнены понятия эффективности и полноты тестирования (результативность/полнота тестовых множеств). На основе полученных данных был проведен сравнительный анализ тестовых методов и разработан алгоритм оптимизации компонентного тестирования.

Практическая значимость данной работы заключается в возможностях применения предложенной методики автоматизации тестирования для оценки программного обеспечения. Теоретические положения исследования вкупе с анализом методов тестирования и подходов легли в основу данной ВКР, основной задачей которой стало является повышение надежности и эффективности ПО. Практическая ценность этого исследования подтверждается результативным использованием разработанной методики и алгоритмов для тестирования программных комплексов различных проектов ООО Энерговектор.

Экспериментальные исследования, проведённые в рамках данного исследования, подтвердили, что предложенные методы тестирования программного обеспечения обладают большей эффективностью по сравнению с традиционными практиками, применяемыми для тестирования программных продуктов информационных систем. Это достигается благодаря использованию подходов, обеспечивающих более высокую скорость выполнения тестов, а также более детальный анализ функциональности программ. Разработанные рекомендации, будучи интегрированными в процесс тестирования, позволяют оптимизировать идентификацию дефектов и повысить общее качество тестируемого программного обеспечения.

Применение более эффективных методов тестирования программного обеспечения имеет ключевое значение, поскольку оно напрямую влияет на качество и надёжность информационных систем, используемых в различных критически важных сферах, таких как здравоохранение, финансы или транспорт. Увеличение быстродействия тестирования позволяет ускорить разработку и выпуск программного обеспечения, что особенно важно в условиях высокой конкуренции на рынке технологий.

Более детальный анализ программ, достигаемый за счёт применения предложенных методов, способствует раннему выявлению дефектов, которые могли бы проявиться в ходе эксплуатации и привести к серьёзным сбоям, включая утрату данных, нарушение конфиденциальности или снижение функциональности системы.

Кроме того, повышение эффективности тестирования минимизирует затраты на исправление ошибок на поздних этапах разработки, что не только оптимизирует расходы, но и повышает доверие пользователей к программным продуктам. Таким образом, внедрение улучшенных методов тестирования обеспечивает не только техническое совершенство программного обеспечения, но и способствует повышению конкурентоспособности и репутации разработчиков.

3.6 Разработка и апробация методики автоматизированного тестирования на базе ООО Энерговектор

На сегодня не существует единого мнения относительно эффективной методологии автоматизированного тестирования ПО. Ниже рассмотрим практическую и технологическую составляющую предложенной нами для ООО Энерговектор методики автоматизированного тестирования. Данная методика применима к различным приложениям, однако здесь мы ее рассмотрим для актуальной платформы 1 С (v8.3).

Ниже представлена диаграмма клиент-серверной архитектуры 1 С v8.3 (рис. 25).

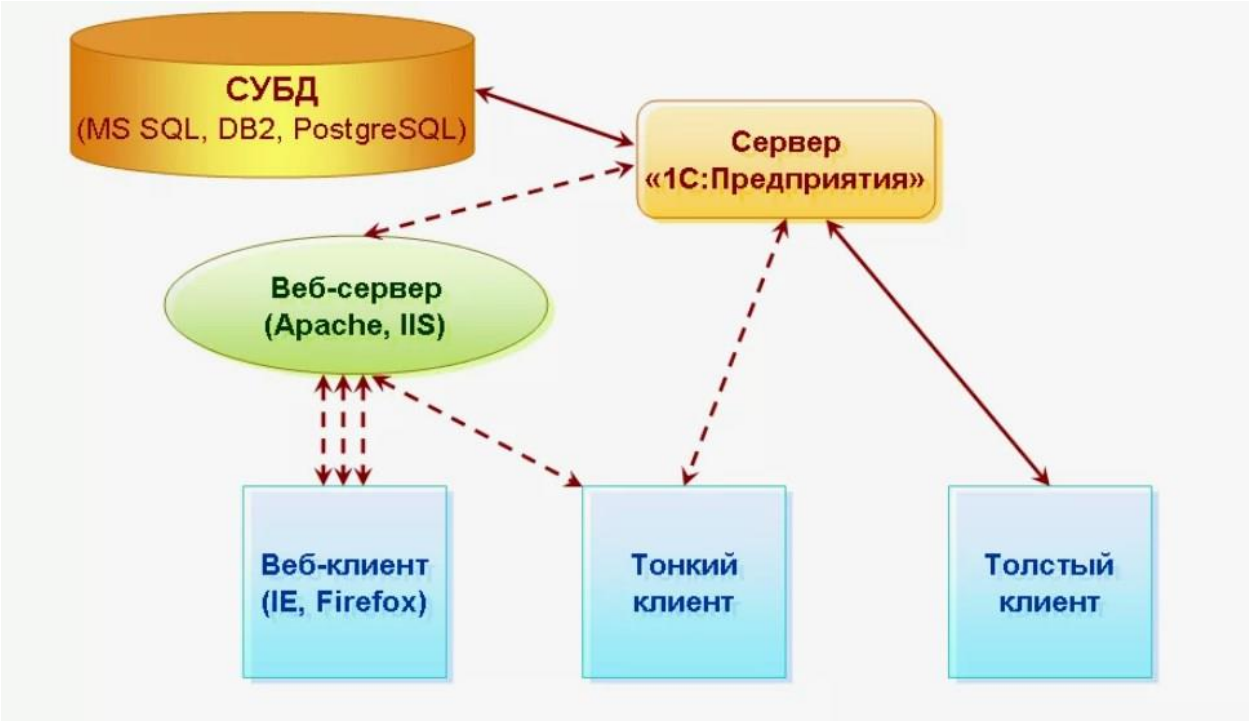

Рисунок 25 -. Диаграмма клиент-серверной архитектуры 1 С (v8.3)

Исходя из данной модели, сформулируем основные условия и требования разрабатываемой методики тестирования.

Среди ключевых требований методики обозначим обязательную проверку в рамках клиент-серверной архитектуры, тестирование производительности и непосредственное применение инструментов автоматизации в зависимости от вида тестирования и обозначенной функциональности объекта тестирования.

Ниже представлена диаграмма управления клиент-серверной архитектурой 1 С v8.3 (рис. 26).

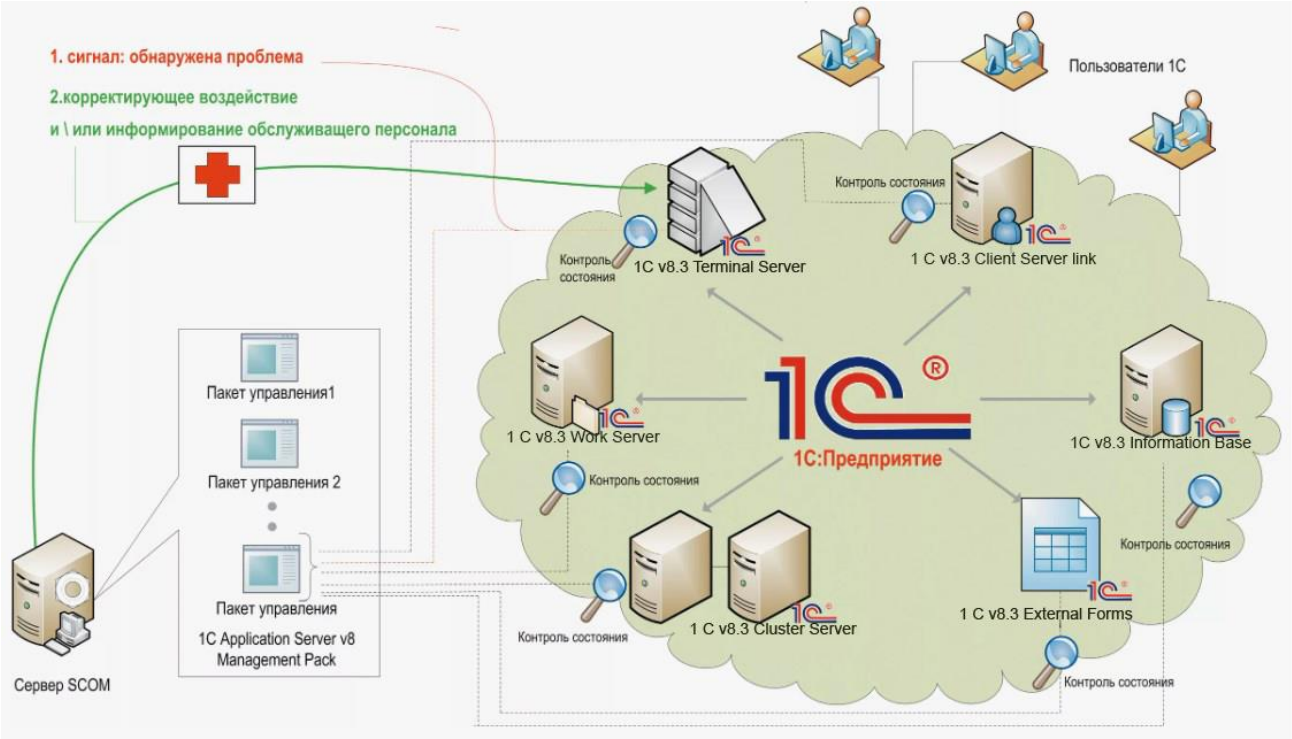

Рисунок 26 - Диаграмма управления клиент-серверной архитектурой 1 С (v8.3)

Существующие на данный момент методики тестирования, как правило, узко ориентированы на архитектуру конкретной платформы, что не позволяет им выйти за пределы тестирования этой платформы, лишая их актуальной универсальности. Поэтому тестировщик, не обладающий навыками и знаниями конкретного платформенного языка, не может провести автоматизированное тестирование рандомной платформы. Существующие методики автоматизированного тестирования предлагают акцентировать внимание на функциональной части, делая упор на модульное и системное тестирование. Задача же нашей методики состоит в разработке универсальной методики комплексного тестирования, в задачи которой входят функциональное, компонентное, API-тестирование, нагрузочное и другие основные виды, на которых мы подробнее остановимся ниже.

Ниже представлены диаграммы "Процессы разрабатываемой методики автоматизированного тестирования для ООО Энерговектор" (рис. 27),

"Жизненный цикл процесса обнаружения и устранения ошибок в ООО Энерговектор согласно разрабатываемой методике" (рис. 28) и структурная диаграмма разрабатываемой методики тестирования (рис. 29), .

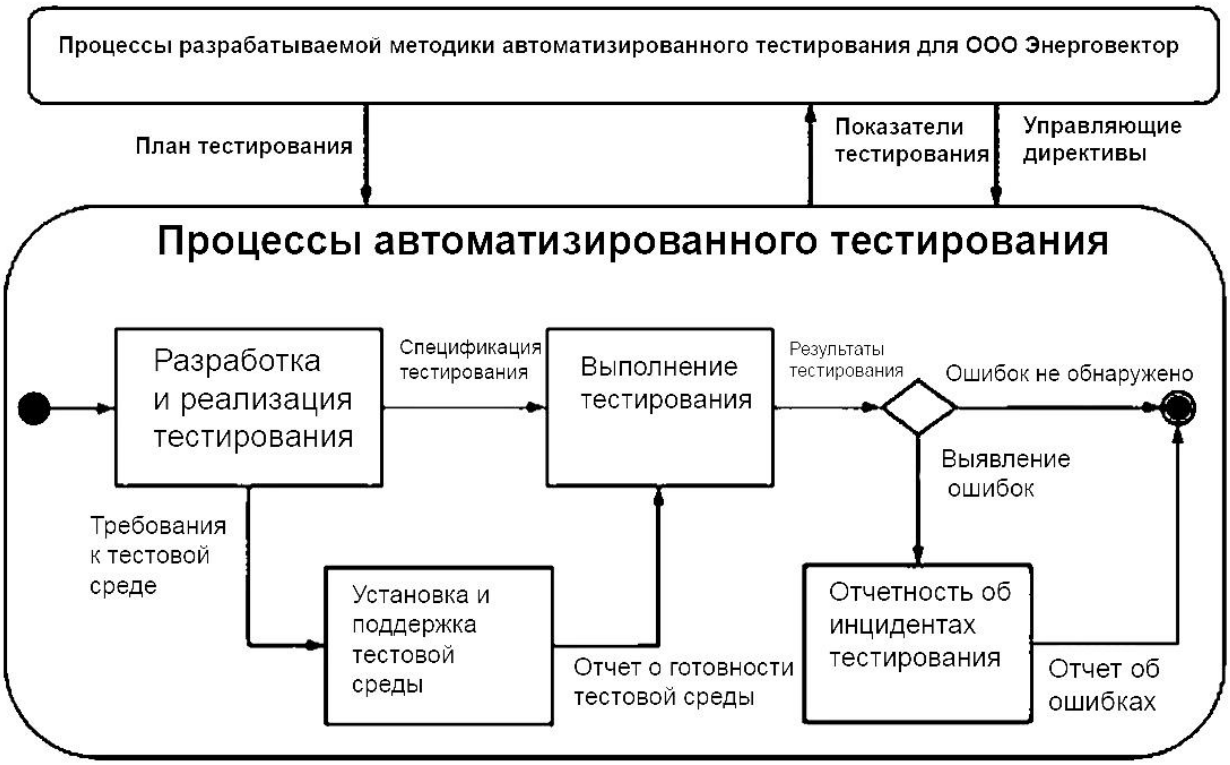

Рисунок 27 - Процессы разрабатываемой методики автоматизированного тестирования для ООО Энерговектор

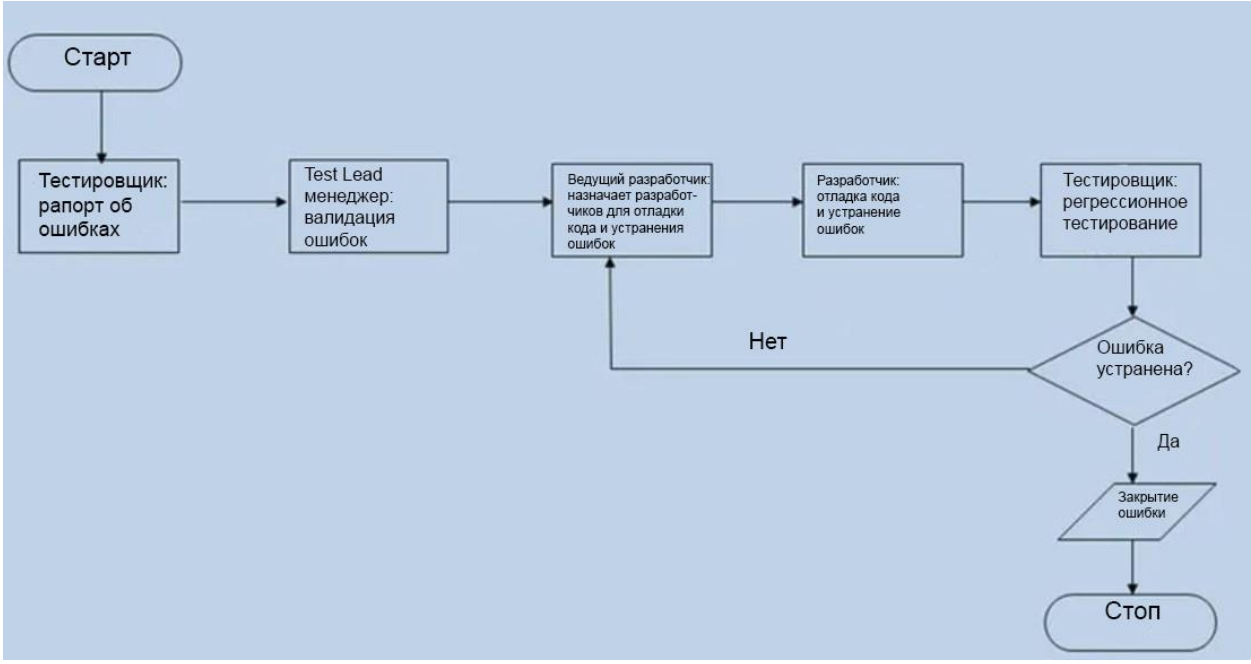

Рисунок 28 - Жизненный цикл процесса обнаружения и устранения ошибок в ООО Энерговектор согласно разрабатываемой методике

Рисунок 29 - Структурная диаграмма разрабатываемой методики тестирования

Апробация разработанной методики будет рассмотрена на примере приложения 1 С (v8.3).

Данная методика включает в себя тестирование конфигурации, серверное тестирование, функциональное, компонентное, нагрузочное и API-тестирование. Ниже представлен визуальный материал, отражающий различные этапы упомянутых видов тестирования согласно разрабатываемой методике. А именно: алгоритм реализации разрабатываемой методики автоматизированного тестирования на базе ООО Энерговектор (рис. 30), старт процесса тестирования, сопровождаемый записью журнала действий (рис. 31), API-тестирование с помощью Runscope (рис. 32) и компонентное тестирование с учетом производительности (рис. 33).

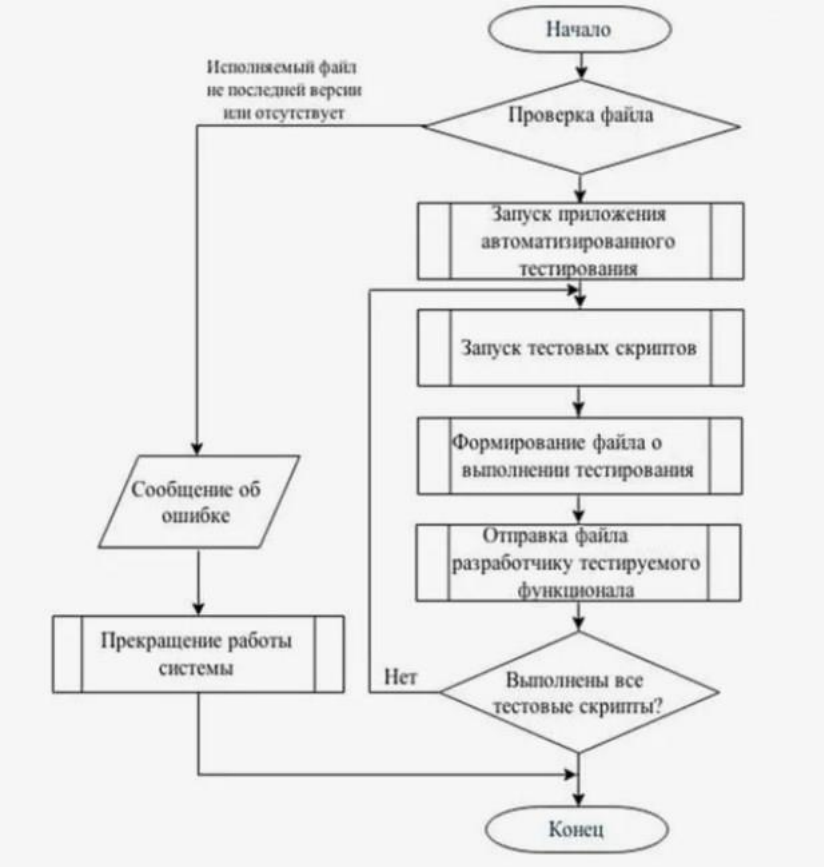

Рисунок 30 -. Алгоритм реализации разрабатываемой методики автоматизированного тестирования (на базе ООО Энерговектор)

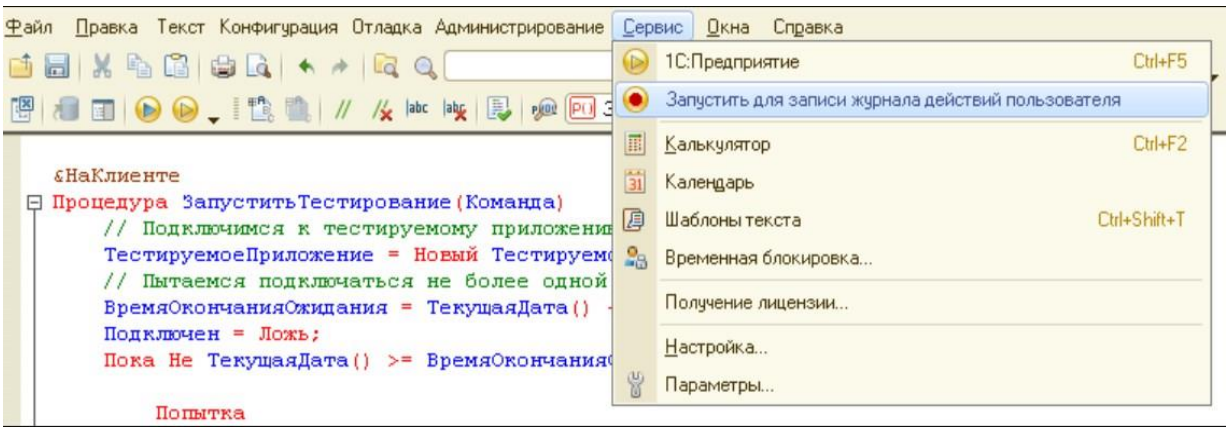

Рисунок 31 - Старт процесса тестирования, сопровождаемый записью журнала действий

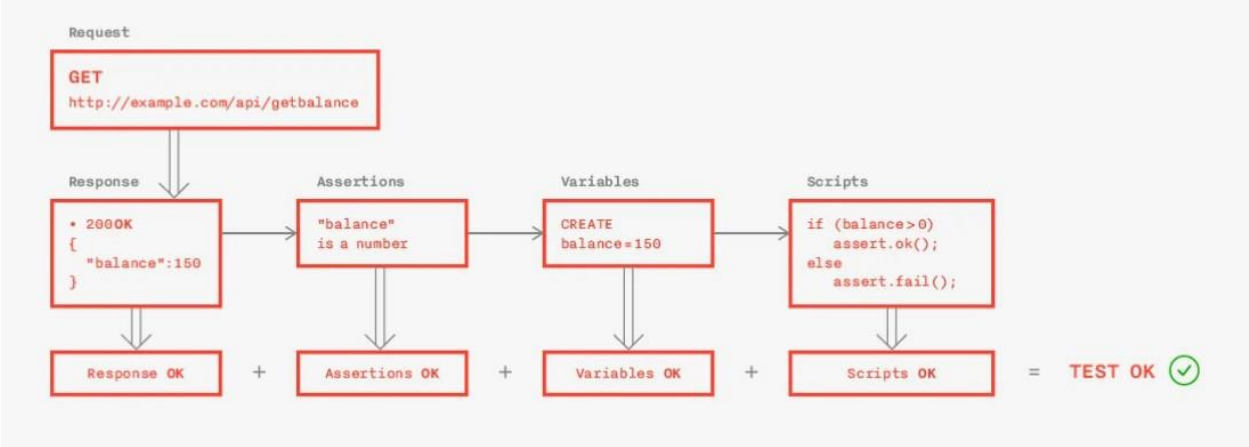

Рисунок 32 - API-тестирование с помощью Runscope

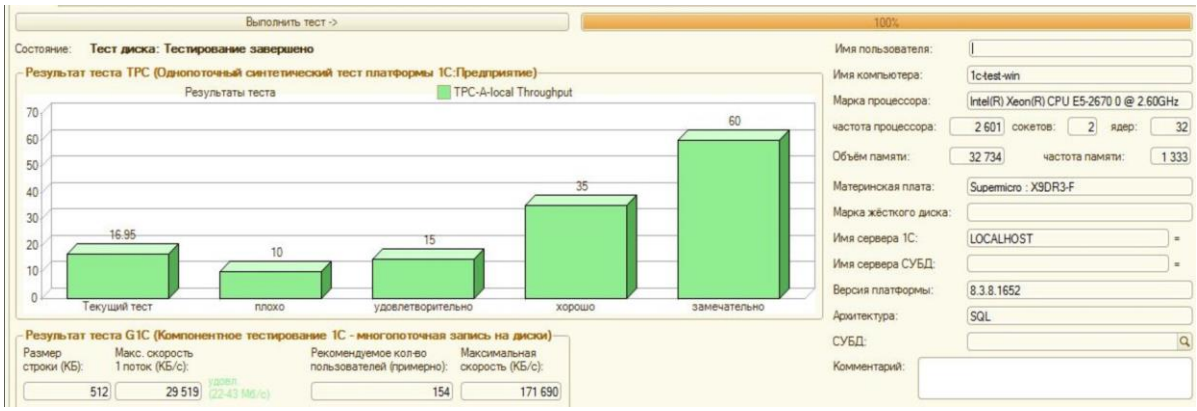

Рисунок 33 - Компонентное тестирование с учетом производительности

При этом задача тестирования конфигурации - проверить совместимость и комбинаторность ПО друг с другом, попутно выяснив оптимальные и наиболее эффективные комбинации из предложенных. В связи с избыточной многозадачностью тестирования конфигурации, принято сужать охват данного тестирования списком конфигураций, которые планируется поддерживать и обслуживать в будущем. Ошибки конфигурации как правило не являются критическими, однако в перспективе приводят к значительному снижению производительности системы. Разработанная нами методика включает блок проверки конфигурации из трех частей: тестовый блок логической адекватности (общий и помодульный), блок отслеживания дефектных ссылок и синтаксический блок. Ниже представлен скриншот окна проверки конфигурации 1С v8.3 (рис. 34).

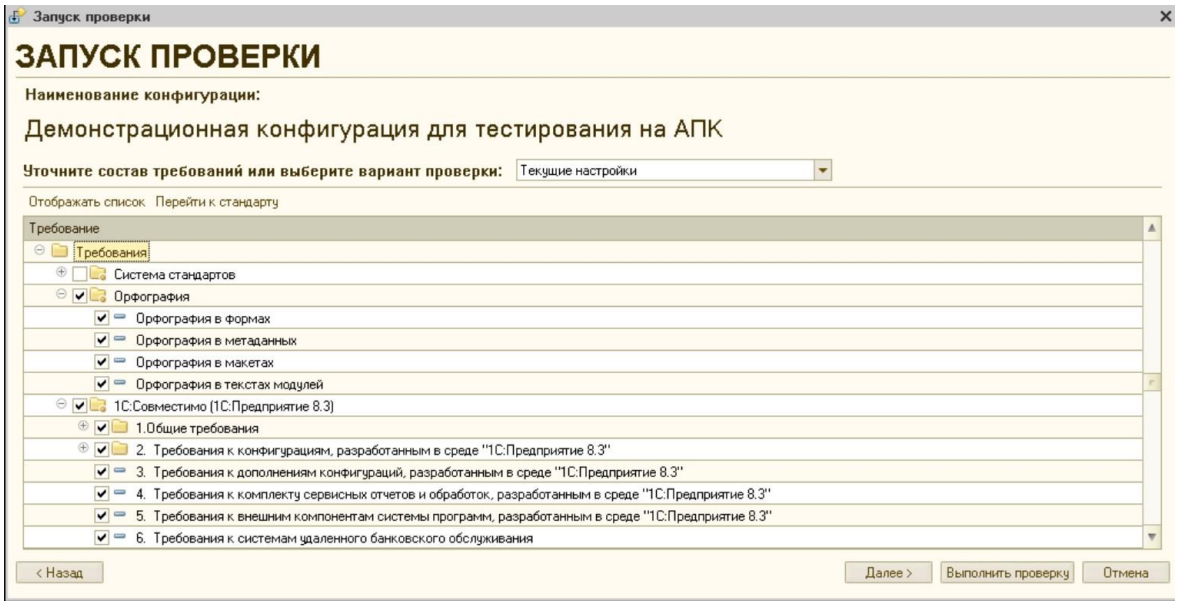

Рисунок 34 - Проверка конфигурации 1С (v8.3)

В рамках нашей методики осуществляется проверка в рамках функциональных, модульных и системных блоков тестирования.

В рамках проверки модулей обычно проверяются вновь созданные, либо же недавно измененные модули. Функциональный блок тестирования является классическим и подразумевает проверку по заданным сценариям. Функциональное тестирование 1 С (v8.3) проще всего произвести средствами автоматизированного тестирования самой платформы.

Блок серверного тестирования нашей методики подразумевает функциональное и нагрузочное тестирование серверов. Для функционального тестирования серверов используется модель тестирования "белого ящика", основной задачей которой является проверка корректности триггеров, запросов, ссылок и остальных компонентов структуры базы данных. Функциональный блок серверного тестирования нашей методики опирается на принципы регрессионного тестирования TDD, подразумевающего тестирование и разработку ПО короткими спринтами, завершающимися процедурой рефакторинга.

Нагрузочное тестирование 1 С (v8.3) проводилось с учетом поддержки PostgreSQL: Tantor SE, Pangolin и Jatoba, Microsoft SQL Server, IBM DB2, Postgres Pro, и Oracle Database. Нагрузочное тестирование разрабатываемой нами методики включает такие блоки, как общий анализ конъюнктуры тестирования, анализ спецификаций, подготовительные системные мероприятия, расчеты возможных конфигураций, обеспечение тестовой среды и оболочки, мониторинг качества тестирования, непосредственно исполнение запланированных сценариев, выгрузка результатов для внесения изменений в код. Нагрузочное серверное тестирование использует тестовую методику Гилева TPC-1C. Суть данного метода сводится к оценке производительности сервера не через оценку его загруженности, но через оценку его производительности за единицу времени. TPC-1C включает блок проверки по компонентам, принципы которого мы разбирали ранее, и т.н. интегральный блок, осуществляющий количественный и качественный анализ производительности системы. Данный тест является универсальным,

кроссплатформенным и как правило используется для анализа скорости и быстродействия однопоточных нагрузок.

Процесс автоматизации тестирования нашей методики включает работу в менеджерах тестирования (таких, как Zephir, ALM Octane, Test IT) и клиентах тестирования. При этом под менеджером тестирования понимаем приложение с соответствующим ключом менеджера, основной задачей которого является исполнение заданных сценариев. В свою очередь клиент тестирования - клиентское приложение с соответствующим ключом клиента, которое выполняет команды менеджера (см. рис. 35).

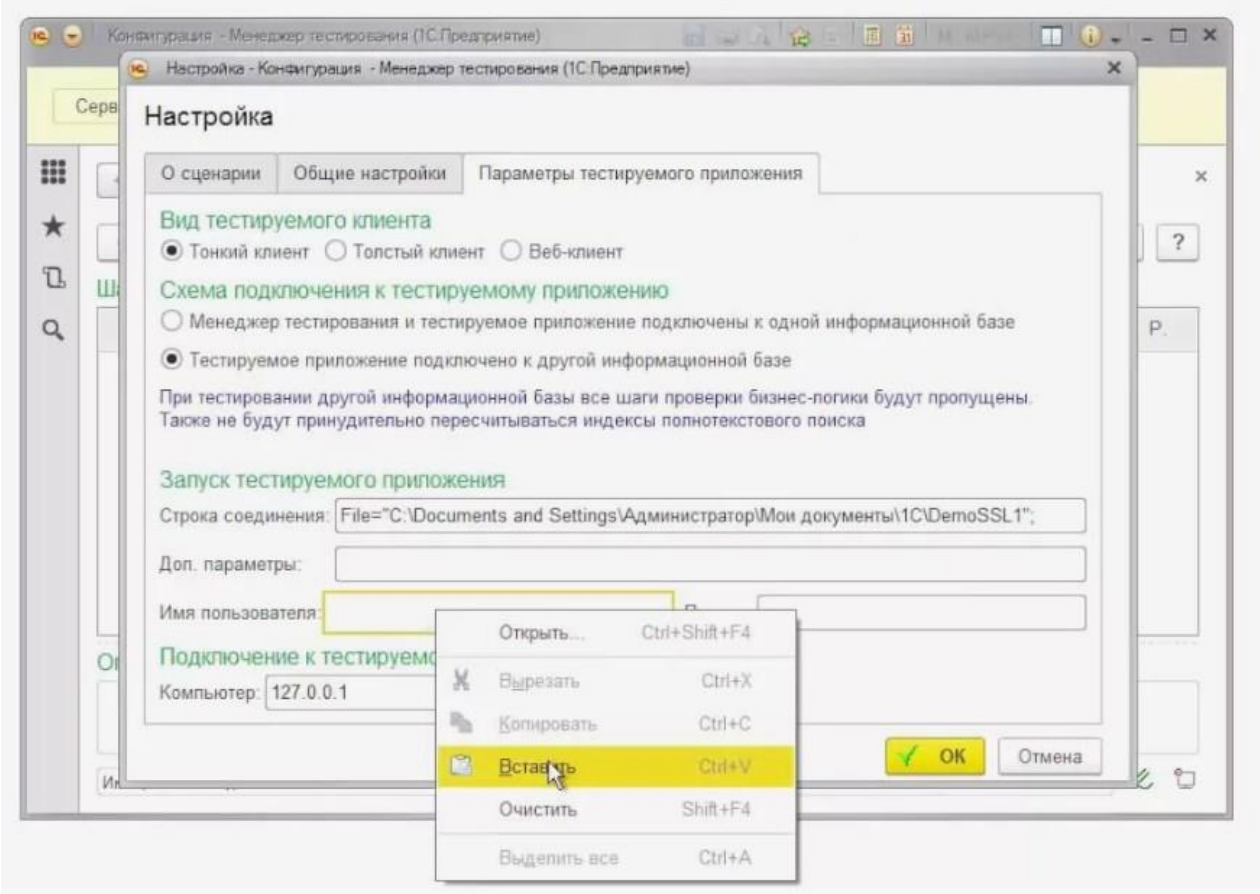

Рисунок 35. - Настройка параметров тестируемого приложения

Наша методика подразумевает два вида запуска этих приложений: 1. запуск на единой базе; 2. запуск на разных базах (для конфигураций «1С v8.3: Сценарное тестирование», «1С v8.3: Тестировщик», «1С v8.3: ERP»).

Если приложения менеджера и клиента находятся на разных базах, то запускать их следует с одинаковой версии платформы. В противном случае они не запустятся, сообщив об ошибке (рис. 36).

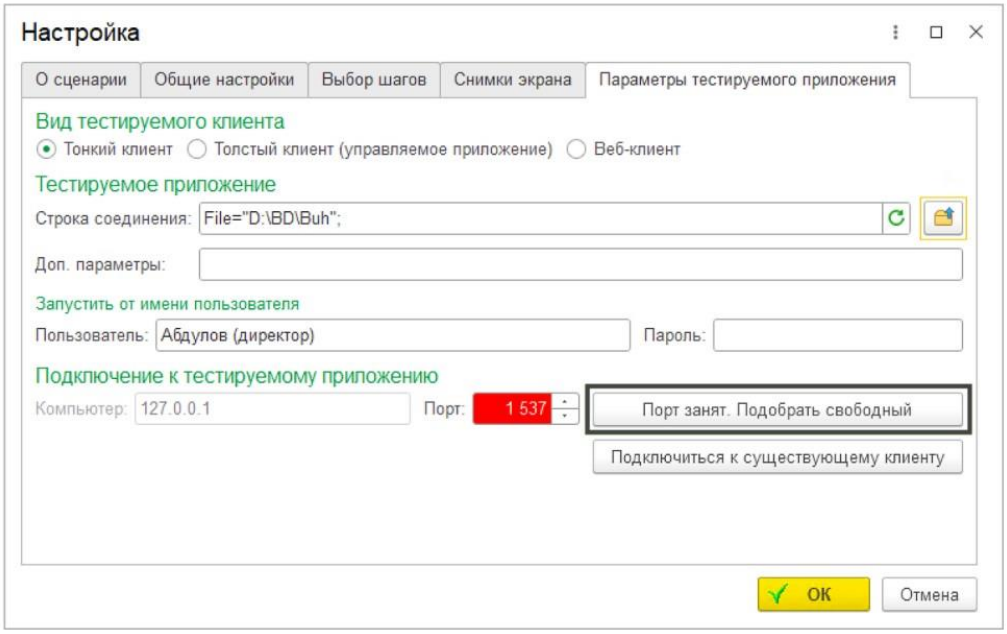

Рисунок 36 - Окно настройки клиент-тестирования

Тестируемое приложения содержит такие основные языковые объекты, как тестируемые приложение, окно, форма, поле формы, группа формы, таблица формы, кнопка формы, интерфейс, группа интерфейса, кнопка интерфейса и др. (см. рис. 37-38).

```
////////////////////////////////////////////////////////////
// *** Выполнить тест - добавить и записать новый товар.

// Открыть форму нового товара.
ГлавноеОкноТестКлиента.ВыполнитьКоманду("e1cib/command/Справочник.Товары.Команда.Создать");

// Если форма нового товара не открылась за 60 секунд, прекратить тестирование.
Если НЕ ТестКлиент.ОжидатьОтображениеОбъекта(Тип("ТестируемаяФорма"), "Товар*") Тогда
    Сообщить("Не удалось открыть форму нового товара в течение 60 секунд.");
    Возврат;

КонецЕсли;

// Получить форму нового товара.
ФормаНовогоТовара = ТестКлиент.НайтиОбъект(Тип("ТестируемаяФорма"), "Товар*");

// Заполнить поле "Наименование".
ПолеНаименование = ФормаНовогоТовара.НайтиОбъект(Тип("ТестируемоеПолеФормы"), "Наименование");
ПолеНаименование.ВвестиТекст("Новый товар (автотест)");

// Заполнить поле "Артикул".
ПолеАртикул = ФормаНовогоТовара.НайтиОбъект(Тип("ТестируемоеПолеФормы"), "Артикул*");
ПолеАртикул.Активизировать();
ПолеАртикул.ВвестиТекст("ПС-0001");

// Записать и закрыть новый товар, нажав на кнопку "Записать и закрыть".
КнопкаЗаписатьИЗакрыть = ФормаНовогоТовара.НайтиОбъект(Тип("ТестируемаяКнопкаФормы"), "Записать и закрыть");
КнопкаЗаписатьИЗакрыть.Нажать();

Сообщить("Тест выполнен.");
```

Рисунок 37 - Сценарий теста создания и записи единицы товара в БД

```
<?xml version="1.0" encoding="UTF-8"?>
<uilog xmlns:d1p1="http://v8.1c.ru/8.3/uilog">
    <ClientApplicationWindow isMain="true">
        <Form title="Электронная почта">
            <close/>
        </Form>
        <Form title="Поиск в данных">
            <close/>
        </Form>
        <Form title="Товары">
            <activate/>
            <FormTable title="Список">
                <FormField title="Наименование">
                    <activate/>
                </FormField>
            </FormTable>
            <FormGroup title="Командная панель">
                <FormButton title="Создать">
                    <click/>
                </FormButton>
            </FormGroup>
        </Form>
    </ClientApplicationWindow>
    <ClientApplicationWindow caption="Товар (создание)">
        <Form title="Товар (создание)">
            <activate/>
            <FormField title="Наименование">
                <inputText text="Новый товар"/>
            </FormField>
            <FormField title="Артикул">
                <activate/>
                <inputText text="123456"/>
            </FormField>
            <FormGroup title="Командная панель">
                <FormButton title="Записать и закрыть">
                    <click/>
                </FormButton>
            </FormGroup>
            <close/>
        </Form>
    </ClientApplicationWindow>
</uilog>
```

Рисунок 38 -Фрагмент тестового сценария обработки формы "Товары"

Ниже приведем результаты отчета тестирования 1 С (v8.3) в рамках разрабатываемой методики.

Таблица 4 - Результаты отчета тестирования 1 С (v8.3) в рамках разрабатываемой методики

Этап	**Результаты**
Проверка конфигурации согласно спецификациям	В пределах нормы
Модульное тестирование	Ошибки устранены
Системное тестирование	Ошибки не выявлены
Межмодульное тестирование	В пределах нормы
Компонентное тестирование	Выявлены ошибки
Тестирование API	В пределах нормы

Исправление ошибок осуществляется через инструмент "Исправление" (рис. 39).

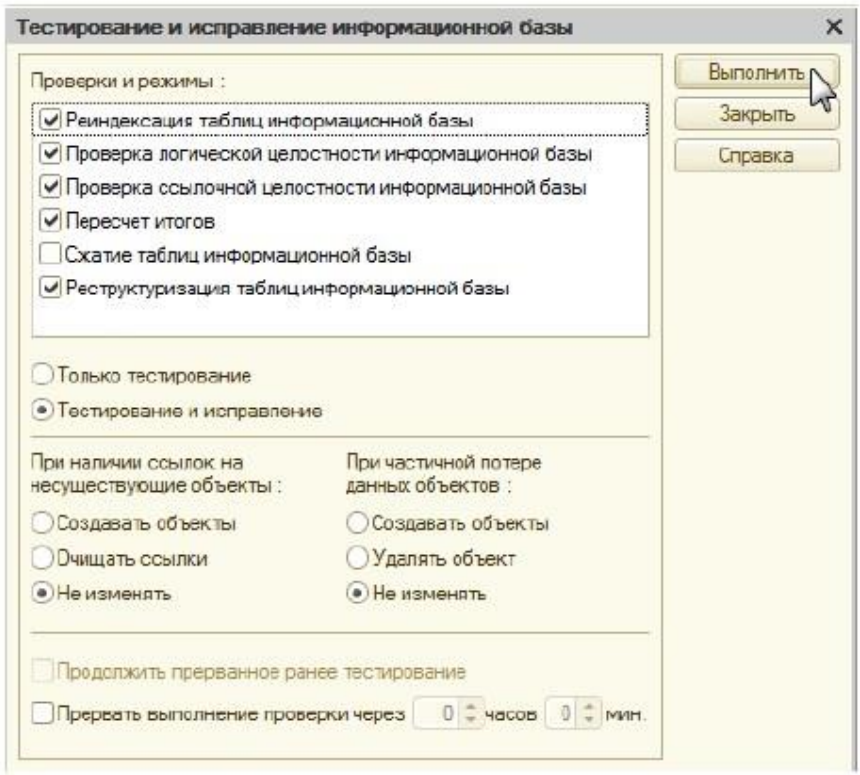

Рисунок 39 - Инструмент "Исправление ошибок" в блоке модульного тестирования

Данный инструмент осуществляет такие задачи, как логический анализ БД, оптимизация и реиндексация с учетом быстродействия, скриннинг корректности элементов и ссылок, сверка результатов, реструктуризация и сжатие заданных объектов и др.

Выводы по третьей главе

Данная методика применима к различным приложениям, однако в работе она рассмотрена для актуальной платформы 1 С (v8.3). Разрабатываемая методика универсальна и предполагает комплексное тестирование, в задачи которого входят тестирование конфигурации, функциональное, компонентное, API-тестирование, серверное, нагрузочное и др. Нагрузочное тестирование 1 С (v8.3) проводилось с учетом поддержки PostgreSQL: Tantor SE, Pangolin и Jatoba, Microsoft SQL Server, IBM DB2, Postgres Pro, и Oracle Database и включает такие блоки, как общий анализ конъюнктуры тестирования, анализ спецификаций, подготовительные системные мероприятия, расчеты возможных конфигураций, обеспечение тестовой среды и оболочки, мониторинг качества тестирования, непосредственно исполнение запланированных сценариев, выгрузка результатов для внесения изменений в код. Нагрузочное серверное тестирование использует тестовую методику Гилева TPC-1C. Процесс автоматизации тестирования нашей методики включает работу в менеджерах тестирования (таких, как Zephir, ALM Octane, Test IT) и клиентах тестирования. Наша методика подразумевает два вида запуска этих приложений: 1. запуск на единой базе; 2. запуск на разных базах (для конфигураций «1С v8.3: Сценарное тестирование», «1С v8.3: Тестировщик», «1С v8.3: ERP»).

Таким образом, разработанная в рамках данного исследования методика автоматизации тестирования была успешно апробирована на платформе 1С (v8.3), показав себя в качестве полезного и эффективного инструмента, возможности которого можно использовать в работе ООО Энерговектор.

Заключение

Результаты исследования. Данное исследование, сосредоточенное на анализе различных методов тестирования программного обеспечения, охватывает подходы, включающие тестирование, не зависящее от спецификаций, ориентированное на спецификации, а также тестирование, связанное с аппаратными средствами. В рамках работы была предпринята попытка разработать методику автоматизированного тестирования, который, основываясь на перечисленных подходах, направлен на улучшение процессов проверки программных компонентов в информационных системах.

В ходе исследования были выделены критерии, обеспечивающие структурированный подход к автоматизированному тестированию. Кроме того, была обозначена спецификация, определяющая параметры и порядок реализации процесса тестирования. Значительное внимание уделено характеристике программных компонентов и их мутаций, включая такие элементы, как ссылки на переменные, операции присваивания, арифметические и логические выражения, которые рассматриваются в контексте различных программных конструкций.

В рамках анализа были подробно исследованы методы тестирования, применимые к ключевым конструкциям, таким как операторы присваивания, условные выражения и циклы, с уточнением их роли в выявлении ошибок и обеспечении стабильности работы программного обеспечения. Также были детализированы понятия эффективности и полноты тестирования, что позволило проанализировать и классифицировать тестовые множества, используемые для проверки корректности работы программных компонентов.

В рамках исследования был проведен сравнительный анализ тестовых методов, разработан алгоритм оптимизации автоматизированного тестирования, проанализированы методы и подходы автоматизации

тестирования ПО. Исследование описывает его практическую ценность, включает диаграммы основных бизнес-процессов с использованием моделей IDEF 0, таких как «Как есть» и «Как должно быть», а также диаграммы декомпозиции, дорожек, потока данных и вариантов использования. Также была представлена модель потока создания ценности и карта пути клиента, а также построена модель мотивации бизнеса (BMM). Определены и описаны точки улучшения бизнес-процессов тестирования, разработан план оптимизации бизнес-процессов для ООО «Энерговектор», назначены ответственные лица и мероприятия, а также установлены сроки оптимизации.

Апробация и возможности внедрения результатов исследования.

В рамках данной научной работы была успешно разработана методика алгоритмов автоматизации тестирования ПО, позволяющая минимизировать затраты времени в 1,5 раза по сравнению с аналогичными конкурентными методиками. Улучшенная методика предполагает автоматическую структуризацию тест-плана и формирование модели на базе спецификаций, которая передает тесты непосредственно в код, исключая при этом сокращение тестового набора.

Практическая ценность проделанной научной работы подтверждается успешным использованием данной методики алгоритмов, позволившей снизить временные затраты почти в 2 раза (до 75 процентов). В рамках исследования на базе ООО Энерговектор были разработаны модули спецификаций, формирования модели и ее оптимизации.

Главной задачей данной ВКР являются оптимизация и повышение эффективности процессов тестирования ПО, снижение временных затрат со стороны инженеров тестирования, а также снижение себестоимости процесса тестирования.

С нашей точки зрения, в обозримом будущем автоматическое тестирование вряд ли сможет полностью заменить ручное, поскольку у автоматического тестирования имеется ряд недостатков таких, как трудоемкость/объемность автоматизации тестовых сценариев и

необходимость непрерывного обновления тестовой базы, что в свою очередь негативно влияет на стоимость процесса тестирования и как следствие - конечную стоимость самого программного продукта.

Вместе с тем автоматическое тестирование имеет ряд неоспоримых преимуществ, среди которых выгодно выделяются скорость и точность применяемых методик автоматизации тестирования, что особенно актуально для крупных IT-компаний, позволяя им за счет грамотного встраивания процессов автоматизации сокращать общие ресурсозатраты на разработку.

Применение методик автоматизации тестирования требует от специалиста значительной компетенции, сравнимой по своему уровню сложности с самой разработкой. Поэтому программисты и тестировщики активно ищут оптимальные подходы для решения возникающих проблем. Настоящее исследование стремится внести свой вклад в поиск решений для обозначенных задач.

Список используемой литературы и используемых источников

1. Александров, Д. В. Инструментальные средства информационного менеджмента. CASE-технологии и распределенные информационные системы : учебное пособие / Д. В. Александров. - Москва, 2022. - 225 с.

2. Артюхова А.С. Проблемы автоматизации тестирования и подходы к их решению// CETERIS PARIBUS. Компьютерные и информационные науки. -2016. -№10.

3. Баженова И.Ю., Сухомлин В.А. Введение в программирование. – М., 2013. – 326 с.

4. 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.

5. Бирюков С.В. Метод и средства автоматизации тестирования интерфейса программирования приложения/ Автореферат диссертации на степень канд. тех. наук. - М., 2011.

6. Блэк Р. Ключевые процессы тестирования. - М., 2017.

7. Богданов, Д.В. Стандартизация процессов обеспечения качества программного обеспечения/ Д.В.Богданов, В.А.Путилов, В.В.Фильчаков. – Апатиты, 1998.

8. Буракова, Ж. А. Анализ хозяйственной деятельности. Практикум : учебное пособие / Ж. А. Буракова, И. В. Карпович, Т. В. Семещенко. - Минск, 2019. - 171 с.

9. Вайнберг Дж. Идеальное программное обеспечение и другие иллюзии в тестировании. - М., 2017.

10. Вайсфельд М. Объектно-ориентированное мышление: Перевод с английского. – СПб, 2014. – 304 с.

11. ГолубевА.М. Методики создания и внедрения агентов в прикладное и системное программное обеспечение для автоматизации тестирования и

мониторинга встроенных вычислительных систем/ Автореферат диссертации на степень канд. тех. наук. - СПб, 2007.

12. Грибанов, Ю. И. Развитие информационной инфраструктуры управления предприятием на основе ИТ-аутсорсинга: монография/ Ю. И. Грибанов, Н. В. Репин, М. Н. Руденко. - Москва, 2019. - 220 с.

13. Гриппа Г.Л. Автоматизация тестирования программных приложений методом ключевых состояний/ Автореферат диссертации на степень канд. тех. наук. - М., СПб, 2006.

14. Дробинцев П.Д. Интегрированная технология обеспечения качества программных продуктов с помощью верификации и тестирования/ Автореферат диссертации на степень канд. тех. наук. - СПб, 2006.

15. Забровский А.Л. Моделирование и автоматизация тестирования процессов передачи мультимедийных потоков на основе комплексной оценки задержек их воспроизведения/ Автореферат диссертации на степень канд. тех. наук. - Петрозаводск, 2013.

16. Золотухина Е.Б. Алфимов Р.В. Красникова С.А. Моделирование предметной области с использованием Enterprise Architect Авторское общество. Свидетельство № 18249 о регистрации произведения результата интеллектуальной деятельности.

17. Кадашев Д. В., Кузнецов А. А. Система распределенного unit-тестирования «Testing GRID»// Вестник НГУ. Том 5, выпуск 1. 2007. С.20-27.

18. Кент Бек. Экстремальное программирование: разработка через тестирование. Библиотека программиста. СПб, 2003. 224с.

19. Колава, Адам; Хейзинга, Дорота (2007). Автоматическое предотвращение дефектов: лучшие практики в управлении программным обеспечением. Wiley-IEEE Computer Society Press. стр. 74.

20. Куликов С. Тестирование программного обеспечения. Базовый курс.

21. Куликова Н.В. Разработка и исследование логических методов тестирования программных комплексов в информационных системах/ Диссертация на звание канд. тех. наук. - М., 2000. - 135 с.

22. Кулянин В., Петренко А., Косачев А., Бурдонов И. Подход UniTesK к разработке тестов// Программирование. 2003. № 29 (6). С. 25-43.

23. Курейчик В.М., Родзин С.И. Компьютерный синтез программных агентов и артефактов// Программные продукты и системы. - 2004. - № 1. - С. 23-27.

24. Лазарев В.А. Анализ и разработка средств интеллектуальной поддержки автоматизированного тестирования программных комплексов/ Автореферат диссертации на степень канд. тех. наук. - Нижний Новгород, 2018.

25. Майерс Г., Баджетт Т., Сандлер К. Искусство тестирования программ, 3-е издание — М., 2012. — 272 с.

26. Макаров А.Н., Малышева О.Н. Моделирование процесса тестирования для автоматизированной системы верификации качества программного обеспечения// Материалы 56-й Научной Конференции БГУИР. - Минск, 2020.

27. Макгрегор Д., Сайкс Д. Тестирование объектно-ориентированного программного обеспечения. Практическое пособие. - К., 2002. - 432 с.

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

29. Маклаков С.В. BPwin, ERwin CASE-средства разработки информационных технологий. – М., 2004 .

30. Мартыненко С., Автоматизированное тестирование с нуля. - 2008.

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

32. Михеева Е.В. Информационные технологии в профессиональной деятельности, 13-е издание: Учебное пособие. – М., 2014. – 384 с.

33. Мищенко А.В. Алгоритмы автоматизации системного тестирования информационных ресурсов/ Автореферат диссертации на степень канд. тех. наук. - Минск, 2019.

34. Мосли Дэниел Дж. Поузи Брюс. Достаточно автоматизации тестирования программного обеспечения. - 2002.

35. Немцова Т.И., Голова С.Ю., Терентьев А.И. Программирование на языке высокого уровня: Учебное пособие – М., 2014. – 511 с.

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

37. Орлов С.А., Цилькер Б.Я. Технологии разработки программного обеспечения: Современный курс по программной инженерии, 4-е издание: Учебник для вузов – СПб, 2012. – 608 с.

38. Плодухин Д.М. Реализация модели автоматизированного тестирования/ Огарев-online 2020. Компьютерные и информационные науки. - М., 2020.

39. Прохорова О.В. Информатика: Учебник. – Самара, 2013. – 106 с.

40. Савин Р. Тестирование dot com. М., 2007.

41. Семакин И.Г., Шестаков А.П. Основы алгоритмизации и программирования, 3-е издание: Учебник. – М., 2012. – 400 с.

42. Симонович С.В. Информатика, 3-е издание: Учебник для вузов – СПб, 2015. – 640 с.

43. Соммервилл И. Инженерия программного обеспечения, 9-ое издание: Перевод с английского. – М., 2011. – 408 с.

44. Уиттакер Д., Арбон Д., Каролло Д., Как тестируют в Google. - СПб, 2014.

45. Уиттакер Дж. Как тестируют в Google. - М., 2014.

46. Хейс, Линда Г. Руководство по автоматизированному тестированию/ Институт тестирования программного обеспечения. - 2004.

47. Черемных С.В., Семенов И.О., Ручкин В.С. Структурный анализ систем: IDEF-технологии. - М., 2001.

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

49. Ambler S.W. and Sadalage P. “Database Refactoring: Evolutionary Database Design”. Boston, 2006.

50. Copeland Lee. A Practitioner's Guide to Software Test Design. - 2003.

51. Elfriede Dustin. Automatad Software Testing. - 1999.

52. Elfriede Dustin. Внедрение автоматизированного тестирования программного обеспечения. - 2009.

53. Garousi V., Mantyla M.V., When and what to automate in software testing? A multi-vocal literature review . - 2016.

54. Hall J.G., Rapanotti L., A design theory for software engineering. - 2017.

55. IEEE Guide to Software Engineering Body of Knowledge. SWEBOK, 2004.

56. Jackvony K. The Complete Software Tester: Concepts, Skills, and Strategies for High-Quality Testing. - 2022.

57. Jacky J., Veanes M., Campbell C., Schulte W. Model-Based Software Testing and Analysis with C#. Cambridge, 2008. 349 p.

58. Li E. “Software Testing in a System Development Process: A Life Cycle Perspective”. Journal of Systems Management, 1990. 41(8). Pp. 23-31.

59. 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.

60. Meyer B. Applying «Design by Contract» // IEEE Computer. 1992. Vol. 25, №. 10. P. 40-51.

61. Monsma J.R. “Model-based testing of Web applications”, Radboud University, 2015.

62. Patton Ron. Software Testing. - 2020.

63. Rodzin S., Rodzina L. Theory of Bioinspired Search for Optimal Solutions and its Application for the Processing of Problem-Oriented Knowledge // В сборнике: 8th IEEE International Conference on Application of Information and Communication Technologies, AICT 2014 Conference Proceedings 8. 2014. С. 7035930

64. Scott Barber R. “Load Models for Performance Testing with Incomplete Empirical Data”. PerfTestPlus, Inc., 2011.

65. Weinberg Gerald M. Perfect Software: And Other Illusions about Testing.

66. Whittaker James A. How to Break Web Software. - 2006.

67. Whittaker James A. How to Break Software Security. - 2004.

68. Williams L. A (Partial) Introduction to Software Engineering Practices and Methods, 2010-2011 (Seventh Edition).: https://online.ist.psu.edu/sites/ist412/files/williamstext.pdf

69. Zarrad A. “A systematic review on regression testing for web-based applications”, 2015, JSW10(8). Pp. 971-990.

70. Zhu H., Hall P. A. V., May J. H. R. Software Unit Test Coverage and Adequacy. ACM Computing Surveys, 29(4). Dec. 1997. Pp. 366-427.