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

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

Кафедра «Прикладная математика и информатика»
(наименование кафедры)
09.04.03 Прикладная информатика
(код и наименование направления подготовки, специальности)

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

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему «Исследование методов комплексного тестирования корпоративной информационной системы»

Студент В.Г. Алексеева _____
(И.О. Фамилия) (личная подпись)

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

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

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

Тольятти 2019

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Интенсивное развитие и все большее внедрение в жизнь достижений информационных технологий (ИТ) требует от проектировщиков и разработчиков информационных систем (ИС) создания эффективных и качественных программных приложений. Это говорит о том, что проверка качества является важным процессом в создании ИС, при этом вопросы и проблемы тестирования освещены в доступной литературе меньше, чем любой другой аспект разработки программного обеспечения.

Анализ публикаций показал, что смысл термина «тестирование» и определение его передается разными авторами как: «тестирование представляет собой процесс, демонстрирующий отсутствие ошибок в программе» [26], «цель тестирования – показать, что программа корректно исполняет предусмотренные функции» [21], «тестирование – это процесс, позволяющий убедиться в том, что программа выполняет свое назначение» [31] и являются недостаточно корректными с точки зрения экономической составляющей (т. е. цены разработки программного продукта (ПП)). Зачастую результаты своей работы программист не может оценить с точки зрения эффективности ИС, в которой она используется, т.е. он практически ничего не может сказать о том, насколько полно протестирован код его программы. Кроме того, очень часто разработчикам, менеджерам проектов, руководителю фирмы необходимо решать вопросы, связанные с сокращением расходов на производство ПП и повышением качества программного обеспечения. Основным способом решения этих проблем является тестирование программного обеспечения ИС. Процесс тестирования включает в себя решение вопросов не только технического характера (организация эффективного процесса тестирования, определение времени тестирования, использования или неиспользования инструментальных средств и т.д.), но и вопросов экономического и психологического характера.

Анализ исследований и публикаций в области тестирования показал, что рассмотрение таких аспектов тестирования, как технические, экономические, психологические представляет значительный практический интерес.

Соответственно вышесказанному, целью является исследование тестов для проверки программных приложений ИС с точки зрения психологического, экономического и технического подходов. Большой вклад в решение различных аспектов проблемы тестирования и программной надежности внесли отечественные ученые: Козлов А.Н., Позин Б.А., Котляров В.П., Липаев В.В., Савин Р., Степанченко И.В., Филиппов В.А., Хатько Е.Е., Шмейлин Б.З., Бородин А.М., Мирвода С.Г., Поршнев С.В., Волков В.Г., Котов С.Л., Кулямин В.В., Петренко А., а также зарубежные специалисты: Andrews A., Offut J., Changyou Xing, Heiskanen H., Maunumaa M., Honlin Zh., Wenbo X., Makinen M. и другие.

Целью диссертационной работы является анализ методов тестирования и разработка сценария комплексного тестирования для повышения производительности и безопасности корпоративных информационных систем.

Обозначенная цель определила объект и предмет исследования.

Объект исследования – методы комплексного тестирования корпоративных информационных систем.

Предмет исследования – разработка сценария комплексного тестирования для повышения производительности и надежности корпоративных информационных систем.

Гипотезой исследования является предположение о возможности создания на основе существующих методов тестирования сценария комплексного тестирования корпоративных информационных систем для повышения их производительности и информационной безопасности. Основываясь на данной гипотезе, применение методов будут наиболее эффективными, если:

- определено понятие комплексного тестирования КИС;
- выбраны методы тестирования, входящие в состав комплексного;
- смоделирован и реализован сценарий проведения комплексного тестирования, позволяющий максимально проверить КИС на ошибки в ее производительности и управлении информационной безопасности.

Для того, чтобы достичь цель необходимо решить задачи:

• рассмотреть теоретические основы тестирования информационных систем, позволяющих осуществлять анализ имеющихся методов тестирования информационных систем;

• проанализировать на основе разработанной теории практические методы тестирования программного обеспечения информационных систем;

• разработать сценарий комплексного тестирования, позволяющего с одной стороны оценить производительность, а с другой – ее информационную безопасность;

• подтвердить путем эксперимента эффективность реализованного сценария и результаты его внедрения, используя процесс апробации.

Научная новизна исследования состоит в том, что в нем определены основные критерии эффективности тестирования, применимые для оценки производительности и надежности КИС; разработан сценарий комплексного тестирования КИС, позволяющий на основе определения наиболее часто встречающихся типов ошибок задавать качественные тесты, обеспечивающие минимизацию всевозможных потерь в бизнес-среде организации.

Практическая значимость исследования заключается в разработке теоретических положений концепции анализа и сравнения методов тестирования информационных систем, позволяющие осуществлять выбор наиболее подходящего метода для практического использования, положенного в основу разработки сценария комплексного тестирования, что позволит существенно сократить затраты на тестирование КИС и повысить качество ее функционирования.

Основные этапы исследования: исследование проводилось с 2017 по 2019 года в несколько этапов:

На первом этапе (констатирующем этапе) – осуществлялся анализ информации по выдвинутой проблеме, на основании которого была сформулирована тема исследования теме исследования, ставилась гипотеза исследования, определялись цель и задачи исследования.

Второй этап (поисковый этап) – осуществлялось моделирование сценария, основанного на методах комплексного тестирования корпоративных информационных систем.

Третий этап (экспериментальная апробация) – Осуществлялась апробация разработанного сценария на нескольких примерах корпоративных систем, что дало уверенность в правильных выводах исследования, проводилась, оценка результатов и были сформулированы выводы о полученном результате по проведенному исследованию.

На защиту выносится:

1. Методы комплексного тестирования корпоративной информационной системы.

2. Разработанный сценарий проведения комплексного тестирования корпоративной информационной системы.

3. Результаты апробации, демонстрирующие эффективность разработанного сценария комплексного тестирования.

В первой главе рассматриваются теоретические основы тестирования корпоративных систем. Во второй главе представлен анализ методов тестирования КИС. В третьей главе приводится процесс апробации реализованного сценария (методики) проведения комплексного тестирования с демонстрацией полученных результатов. Четвертая глава посвящена процессу оценки эффективности реализованного сценария комплексного тестирования КИС.

В заключении подводятся итоги выполненной работы.

Диссертация состоит из введения, четырех глав, заключения, списка литературы. Работа изложена на 113 страницах и содержит 63 рисунка.

ГЛАВА 1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ТЕСТИРОВАНИЯ КОРПОРАТИВНЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

1.1 Жизненный цикл тестирования корпоративной информационной системы

Корпоративные информационные системы (КИС) с точки зрения системного анализа относятся к сложным системам, поскольку они построены из множества различных подсистем: серверов БД, серверов приложений, кэширующих серверов, балансировщиков, серверов резервного копирования, безопасности и т.д. [18]. В хорошо спроектированной системе велика степень взаимной интеграции, которая увеличивает риск выхода из строя одной или нескольких подсистем КИС, за которым следует выход из строя других ранее работоспособных подсистем и так далее вплоть до полного разрушения всей КИС. Без специальных средств обработки нештатных ситуаций КИС окажется неустойчивой к ошибкам и не сможет гарантировать заложенные в технические требования и соглашение об уровне услуг требования по времени восстановления до работоспособного состояния в случае возникновения предвиденных и непредвиденных ошибок [11].

В этой связи разработка методик (сценариев) тестирования КИС, проводимого на всех этапах ее жизненного цикла, является актуальной [8]. *Жизненный цикл тестирования программного обеспечения (STLC)* - это процесс тестирования, который выполняется систематически и в плановом порядке. В процессе STLC для улучшения качества продукта выполняются различные действия. В жизненном цикле тестирования программного обеспечения (STLC) предусмотрены следующие этапы со своими критериями входа и результатами:

1. Анализ требований (табл. 1.1) - на этом этапе команда обеспечения качества (QA) понимает требования с точки зрения того, что мы будем тестировать, и выясним тестируемые требования. Требования могут быть функциональными или нефункциональными, такими как производительность, тестирование безопасности [10].

Таблица 1.1 – Критерии входа и результата этапа «Анализ требований»

Критерии входа	Мероприятия	Результат
Следующие документы должны быть доступны: - Технические требования. - Приложение архитектурное Наряду с вышеуказанными документами критерии приемки должны быть четко определены.	Подготовьте список вопросов или запросов и получите ответы от Business Analyst, System Architecture, Client, Technical Manager / Lead и т.д. Составьте список того, что выполняли все типы тестов, такие как функциональность, безопасность, производительность и т.д. Определите направление и приоритеты тестирования. Перечислите детали среды тестирования, где будут проводиться действия по тестированию. При необходимости проверьте технико-экономическое обоснование автоматизации и подготовьте технико-экономическое обоснование автоматизации.	Список вопросов со всеми ответами, которые должны быть решены из бизнеса, т.е. тестируемые требования ТЭО автоматизации (если применимо)

2. Планирование испытаний (табл. 1.2) – на этом этапе определяются усилия и оценки затрат для всего проекта.

Таблица 1.2 – Критерии входа и результата этапа «Планирование испытаний»

Критерии входа	Мероприятия	Результат
Документы с требованиями (обновленная версия неясного или отсутствующего требования). Автоматизация технико-экономического обоснования.	Определить цель и объем проекта. Перечислите типы тестирования, участвующие в STLC. Оценка усилий по тестированию и планирование ресурсов. Выбор инструмента тестирования, если требуется. Определите обзор процесса тестирования. Определите среду тестирования, необходимую для всего проекта. Подготовьте расписание испытаний. Определите контрольные процедуры. Определение ролей и обязанностей. Перечислите результаты тестирования. Определите критерии входа, критерии приостановки, критерии возобновления и критерии выхода. Определите риск, если таковой имеется.	План тестирования или документ о стратегии тестирования. Документ по оценке усилий по тестированию.

3. Разработка тестового примера (таб. 1.3) – на этом этапе STLC команда тестирования записывает подробные тестовые примеры. Наряду с тестовыми

наборами команда тестирования также готовит тестовые данные, если таковые необходимы для тестирования.

Таблица 1.3 – Критерии входа и результата этапа «Разработка тестового примера»

Критерии входа	Мероприятия	Результат
Документы с требованиями (обновленная версия неясного или отсутствующего требования). Автоматизация технико-экономического обоснования.	Подготовка тестовых случаев. Подготовка сценариев автоматизации тестирования (при необходимости). Подготовка необходимых тестовых данных для выполнения тестовых случаев.	Тестовые случаи. Тестовые данные. Тестирование сценариев автоматизации (если требуется).

4. Настройка тестовой среды (табл. 1.4), которая определяет условия тестирования программного обеспечения.

Таблица 1.4 – Критерии входа и результата этапа «Настройка тестовой среды»

Критерии входа	Мероприятия	Результат
План испытаний доступен. Тесты дыма доступны. Тестовые данные доступны.	Проанализируйте требования и подготовьте список программного и аппаратного обеспечения, необходимого для настройки тестовой среды. Настройте тестовую среду. После настройки среды тестирования выполните контрольные тесты, чтобы проверить готовность среды тестирования.	Тестовая среда будет готова с тестовыми данными. Результат тестов на дым.

5. Выполнение теста (табл. 1.5) – на этом этапе команда тестирования начинает выполнение тестовых примеров на основе подготовленного планирования тестов и подготовленных тестовых примеров на предыдущем этапе. Если какой-либо из тестовых случаев заблокирован из-за какого-либо дефекта, то такие тестовые случаи могут быть помечены как заблокированные, поэтому мы можем получить отчет на основе того, сколько тестовых примеров прошло, провалилось, заблокировано или не выполнено и т.д. После устранения

дефектов, те же тесты с ошибками или блокировка могут быть выполнены снова для повторного тестирования функциональности [10].

Таблица 1.5 – Критерии входа и результата этапа «Выполнение теста»

Критерии входа	Мероприятия	Результат
План тестирования или документ о стратегии тестирования. Тестовые случаи. Тестовые данные.	На основе планирования тестирования выполните контрольные примеры. Отметьте состояние тестовых случаев, таких как Пройдено, Сбой, Заблокировано, Не выполнено и т.д. Назначьте идентификатор ошибки для всех неудачных и заблокированных тестовых случаев. Сделайте повторное тестирование, как только дефекты устранены. Отслеживайте дефекты до закрытия.	Отчет о выполнении теста. Отчет о дефектах.

6. Закрытие цикла испытаний (табл. 1.6) – на этом этапе анализируются ошибки, распределяются дефекты по типу и сложности.

Таблица 1.6 – Критерии входа и результата этапа «Закрытие цикла испытаний»

Критерии входа	Мероприятия	Результат
Выполнение тестового примера завершено Отчет о выполнении теста Отчет о дефектах	Оцените критерии завершения цикла на основе охвата тестирования, качества, стоимости, времени, критических бизнес-целей и программного обеспечения. Подготовьте показатели теста на основе вышеуказанных параметров. Подготовить отчет о закрытии теста Поделиться передовым опытом для любых подобных проектов в будущем	Отчет о закрытии теста Тест метрики

Сценарии тестирования STLC [18] должны способствовать поиску непредвиденных ошибок/проблем в работе КИС на всех этапах жизненного цикла, который выражается замкнутой последовательностью действий (рис. 1.1):

- стадия 1 - общее планирование и анализ требований – для определения методов и видов тестирования, ограничений и рисков, предстоит тестировать; наличия необходимого инструментария и т.п.;

- стадия 2 - уточнение критериев приёмки – для определения/уточнения метрик и признаков возможности/необходимости начала, приостановки и возобновления тестирования, завершения или прекращения тестирования;
- стадия 3 - уточнение стратегии тестирования - для рассмотрения и уточнения актуальных для текущей итерации частей стратегии тестирования;
- стадия 4 - разработка тест-кейсов – для разработки, пересмотра, уточнения, доработки, переработки и т.п. с тест-кейсами/тестовыми сценариями;
- стадия 5 - выполнение тест-кейсов – для непосредственного выполнения сценария тестирования;
- стадия 6 - фиксация найденных дефектов - для формирования понимания проблемы и уточнения важности и срочности проведения тестирования;
- стадия 7 - анализ результатов тестирования – для обработки полученных результатов, чтобы принимать решение об еще одной итерации;
- стадия 8 – отчетность - для фиксирования промежуточных и конечных вариантов тестирования.

Рисунок 1.1 – Жизненный цикл тестирования

Каждая из выделенных стадий тестирования может рассматриваться как обеспечивающая создание определенного промежуточного продукта –

функциональной группы программ или программного средства с некоторыми ограниченными характеристиками качества. Эти характеристики выделяются и детализируются на основе первичного технического задания и спецификации требований на ПС. В процессе проектирования ПС они уточняются и конкретизируются в спецификациях требований на группы программ и их компоненты [20]. В результате создается совокупность эталонов, имеющих последовательно расширяющиеся номенклатуру и наборы значений показателей качества, которым должны соответствовать отлаживаемые и испытываемые компоненты на каждой стадии тестирования.

Таким образом, пройдя основные стадии жизненного цикла тестирования ИС можно с уверенностью утверждать о работоспособности программного продукта.

1.2 Принципы и основные этапы комплексного тестирования корпоративной информационной системы

Тестирование КИС охватывает основные стадии жизненного цикла, аналогичный последовательности процессов разработки программного обеспечения: постановка задачи для теста, проектирование, написание тестов, тестирование тестов, выполнение тестов и изучение результатов тестирования [22]. Чтобы ориентироваться в тестах, стоит рассмотреть два основных подхода.

Первый подход ориентирован на изучение логики программного обеспечения КИС при проектировании тестов. В процессе проектирования тестов необходимо предусмотреть, чтобы каждая команда условного перехода выполнялась в каждом направлении хотя бы раз, т.е. нужно проверить каждую ветвь алгоритма, определяющую некоторый путь, [14]. При этом совсем (или почти совсем) не интересуются спецификациями.

Второй подход тестирования основан на выполнении фундаментального принципа функционирования КИС: отдача должна превышать затраты. Эта отдача измеряется вероятностью того, что тест выявит необнаруженную прежде ошибку. Затраты измеряются временем и стоимостью подготовки, выполнения и

проверки результатов теста [16]. Каждый тест должен быть представлен некоторым классом входных значений таким образом, чтобы его правильное выполнение создавало убежденность в том, что программа будет выполняться правильно для определенного класса входных данных.

Если учесть, что КИС – это не только используемые в ее составе программные компоненты, но и аппаратное и организационное обеспечение, то и в результатах ее испытаний должны быть отражены показатели выбранных серверов, рабочих станций, сетевого оборудования (их надежность и производительность), а также эффективность разработанного регламента эксплуатации системы. В связи с этим возникает необходимость в проведении комплексного тестирования на соответствие всем предъявляемым требованиям.

Комплексное тестирование КИС состоит из следующих этапов [33]:

1. Подробное изучение проекта системы и её эксплуатационных документов.

2. Создание и внедрение автоматизированной системы отслеживания ошибок (bug tracking).

3. Непосредственное тестирование работы ИС, которое включает в себя:
   1) тестирование работы всех модулей ИС,
   2) проверка внутрисистемных связей,
   3) тестирование оборудования, в том числе на наличие необходимых лицензий,
   4) проверка программного обеспечения, в том числе проверка кода,
   5) тестирование производительности и максимальных нагрузок ИС,
   6) тестирование на отказы: неожиданные программные сбои, выход из строя модулей системы, человеческий фактор и др.,
   7) тестирование на защищенность ИС — включает в себя комплекс исследований по выявлению способов взлома системы и утечек информации,
   8) общая проверка работы системы.
4. Тестирование работы системы управления it-структурой.

5. Тестирование персонала.

6. Анализ полученных данных и выработка стратегии по исправлению выявленных ошибок.

Комплексное тестирование не предназначено для тестирования всех функций полностью собранной системы. Оно направлено на поиск несоответствия системы ее исходным целям [34]. Таким образом, в комплексном тестировании принимаю участие КИС, описание ее целей и вся документация, которая будет поставляться вместе с системой.

1.3 Методология тестирования корпоративных информационных систем

Методология тестирования является стратегией и подходом для тестирования ИС, чтобы гарантировать, что программный продукт пригоден для эксплуатации.

Методики тестирования включают тестирование того, что ИС работает в соответствии с спецификацией, не имеет нежелательных побочных эффектов при использовании способами, выходящими за пределы проектных параметров, и в худшем случае будет отказоустойчивым [27].

Методологии тестирования ИС - это различные подходы и способы обеспечения того, чтобы ИС была полностью протестирована. Методологии тестирования охватывают все: от модульного тестирования отдельных модулей, интеграционного тестирования всей ИС до специализированных форм тестирования, таких как безопасность и производительность [11].

Поскольку ИС становятся все более сложными и взаимосвязанными, а также с большим количеством различных платформ и устройств, которые необходимо протестировать, важно иметь надежную методологию тестирования, чтобы убедиться, что разрабатываемые программные продукты / ИС были полностью протестированы, что они соответствуют указанным требованиям и могут успешно работать во всех ожидаемых средах с требуемым удобством использования и безопасностью.

На рис. 1.2 представлена общая схема основных видов тестирования ИС, предусмотренных методологиями тестирования.

Рисунок 1.2 – Общая схема основных видов тестирования

Согласно данной схеме методология тестирования включает [36]:

1. Функциональное тестирование выполняется с использованием функциональных спецификаций, предоставленных клиентом, или с использованием спецификаций проектирования, таких как сценарии использования, предоставляемые командой разработчиков.

Функциональное тестирование в методологии тестирования разбито на четыре компонента: модульное тестирование, интеграционное тестирование, системное тестирование и приемочное тестирование.

Модульное тестирование - тестирование отдельных программных модулей или компонентов, входящих в состав приложения или системы. Модульные тесты обычно пишутся разработчиками модуля, и в методологии разработки, основанной на тестировании (такой как Agile, Scrum или XP), они фактически пишутся до того, как модуль будет создан как часть спецификации [19]. Каждая функция модуля тестируется специальным модульным тестовым прибором, написанным на том же языке программирования, что и сам модуль (рис. 1.3).

```
public class SampleTestFixture
{
        [SetUp]
        public void Init()
        {
                //Do Nothing
        }

        /// <summary>
        /// Sample test that asserts a failure
        /// </summary>
        [
        Test,
        SpiraTestCase (<test case id>)
        ]
        public void _01_SampleFailure()
```

Рисунок 1.3 – Фрагмент модульного теста

Интеграционное тестирование - тестирование различных модулей / компонентов, которые были успешно протестированы при интегрировании вместе для выполнения конкретных задач и мероприятий (также известные как тестирование сценария). Это тестирование обычно выполняется с помощью комбинации автоматических функциональных тестов и ручного тестирования.

Системное тестирование включает тестирование всей системы на наличие ошибок. Данный вид теста проводится путем взаимодействия с аппаратными и программными компонентами всей системы, а затем осуществляется проверка в целом (рис. 1.4). Для данного типа тестирования применяется метод «черного ящика», где программное обеспечение проверяется на наличие ожидаемых рабочих условий, а также возможных исключительных и граничных условий.

Name	Execution Status	Planned Date	Release	Last Executed	Owner	Status	ID	Edit
	-- Any --		-- Any --		-- Any --	-- Any --	TX	Edit
Exploratory Testing 2					Fred Bloggs	Deferred	TX:6	Edit
Testing Cycle for Release 1.0 7		4-Feb-2007	1.0.0.0	1-Dec-2003	Joe P Smith	In Progress	TX:1	Edit
Testing Cycle for Release 1.1 9		6-Feb-2007	1.1.0.0	1-Dec-2003	Joe P Smith	Not Started	TX:2	Edit
Testing New Functionality 4		9-Feb-2007	1.2.0.0		Fred Bloggs	In Progress	TX:5	Edit

Show 15 rows per page Displaying page 1 of 1

Рисунок 1.4 – Демонстрация работы системного тестирования

Приемочное тестирование (рис. 1.5) является заключительным этапом функционального тестирования программного обеспечения и включает в себя проверку того, что все требования к продукту выполнены, и что конечные пользователи и клиенты протестировали систему, чтобы убедиться, что она работает должным образом и отвечает всем предъявленным требованиям [38].

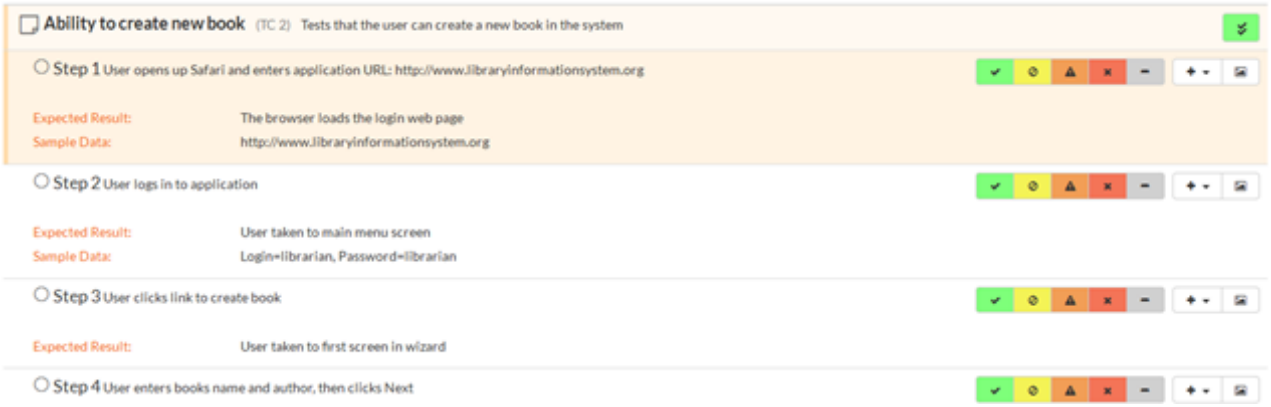


Рисунок 1.5 – Демонстрация работы приемочного тестирования системы

2. Нефункциональное тестирование включает в себя тестирование приложения на соответствие нефункциональным требованиям, которые обычно включают измерение / тестирование приложения на соответствие определенным техническим качествам («способностям»), например: уязвимость, масштабируемость, удобство использования [11, 24].

В большинстве методологий тестирования существует несколько различных типов тестирования производительности (рис. 1.6), например [27]:

- тестирование производительности - это измерение поведения системы при возрастающей нагрузке (как числа пользователей, так и объемов данных),
- нагрузочное тестирование - проверка того, что система может работать в требуемом режиме. Время отклика, когда он подвергается ожидаемой нагрузке,
- стресс-тестирование - это определение точки отказа в системе, когда протестированная нагрузка превышает ту, которую она может поддерживать.

Рисунок 1.6 – Демонстрация работы тестирования производительности

Тестирование совместимости (рис. 1.7) проверяет совместимость продукта или приложения со всеми указанными операционными системами, аппаратными платформами, веб-браузерами, мобильными устройствами и другими разработанными сторонними программами (например, плагинами браузера). Тесты совместимости нужны [31], чтобы убедится, что программный продукт работает, как ожидается, во всех различных комбинаций аппаратных / программных, и что все функции последовательно поддерживаются.

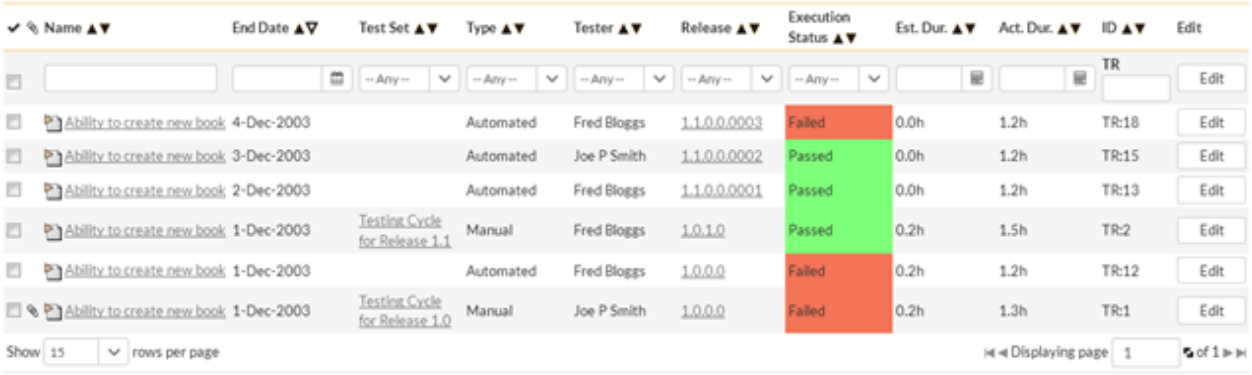


Рисунок 1.7 – Демонстрация работы тестирования совместимости

Тестирование безопасности проверяет программное обеспечение на конфиденциальность, целостность, аутентификацию, доступность и неприкосновенность. Индивидуальные тесты проводятся для предотвращения любого несанкционированного доступа к коду программного обеспечения.

Юзабилити-тестирование рассматривает аспект удобства использования программного обеспечения для конечного пользователя. Простота, с которой пользователь может получить доступ к продукту, формирует основную точку тестирования. Юзабилити-тестирование рассматривает пять аспектов тестирования: обучаемость, эффективность, удовлетворенность, запоминаемость и ошибки.

Таким образом, совокупность рассмотренных тестовых испытаний может быть представлена в виде комплексного тестирования (integration tests), проводимого во время процесса интеграции технического и программного обеспечения до валидации компьютерной системы с целью проверки совместимости программного обеспечения и технического обеспечения компьютера.

Все комплексные тесты должны быть подготовлены на основе документации для пользователя. Они пишутся в форме сценариев, представляющих ряд последовательных действий пользователям и состоят из трех основных компонентов [37]:

1) собственно сценария комплексного тестирования с указанием действий, которые должны быть совершенны во время выполнения теста

2) входных данных,

3) ожидаемых выходных данных.

В комплексном тестировании помимо специалистов могут принимать участие и пользователи КИС, основываясь на одном из методов [42]:

1. Опытная эксплуатация, при которой система тестируется на рабочем месте. Это позволяет увидеть КИС в работе до того, как ее начнут эксплуатировать.

2. Использование КИС в организации изготовителя для внутренних нужд. Позволяет устранить множество ошибок, но не позволяет увидеть некоторые ошибки интеграции.

Таким образом, комплексное тестирование КИС может быть процессом, как контроля, так и испытания, при котором можно увидеть ее работу в реальной

среде пользователя или в обстановке, которая создана, чтобы максимально эмулировать среду пользователя.

1.4 Виды тестов при выполнении комплексного тестирования корпоративных информационных систем

Компонентами комплексного теста являются исходные цели, документация, публикации для пользователей и сама система. Все комплексные тесты должны быть подготовлены на основе публикаций для пользователя (а не внешних спецификаций). К внешним спецификациям следует обращаться только для того, чтобы разобраться в противоречиях между системой и публикациями о ней [43]. Выделяют следующие подвиды комплексного тестирования:

1. *Тестирование стрессов или тестирование с нагрузкой* - попытка подвергнуть систему максимальному «давлению» (например, попытку одновременно подключить к системе разделения времени 100 терминалов).

2. *Тестирование объема* - попытка предъявить системе большие объемы данных в течение длительного времени, чтобы определить соответствие количества обрабатываемых данных и данных, указанных в спецификации.

3. *Тестирование конфигурации* – попытка проверить работы минимальной и максимальной конфигурации системы, с любой аппаратной платформой или программой, с которой КИС будет взаимодействовать.

4. *Тестирование совместимости* – попытка найти несовместимости новых и старых версий КИС, при этом взаимодействие пользователей с прежней версией должно полностью сохраниться.

5. *Тестирование защиты* – попытка нарушить секретность в системе.

6. *Тестирование требований к памяти* – попытка показать, что система не достигает тех целей в памяти, которые прописаны в сопровождающей документации.

7. *Тестирование производительности* – попытка продемонстрировать, что данная программа не соответствует заявленным характеристикам, таким как: время отклика, уровень пропускной способности при определенной нагрузке.

8. *Тестирование процессов настройки системы* – попытка определить качество и эргономичность работы системы.

9. *Тестирование надежности/готовности* - попытка доказать, что система не удовлетворяет исходным требованиям надежности (среднее время между отказами, количество ошибок, способность к обнаружению, исправлению ошибок и устойчивость к ошибкам).

10. *Тестирования средств восстановления* – попытка проверить способность системы к восстановлению (особенно в области работы операционной системы, СУБД, систем передачи данных).

11. *Тестирование удобства обслуживания* – попытка проанализировать глазами обслуживающего персонала документы, описывающие внутреннюю логику системы (требования к продукту, к проекту, определяющие удобства обслуживания (сопровождение системы)), чтобы понять, как быстро и точно указать причину ошибки, если известны только некоторые ее симптомы.

12. *Тестирование публикаций* – попытка поверить точность всей документации.

13. *Тестирование психологических факторов* – попытка устранения мелких недостатков, минимизация неудобства использования.

14. *Тестирования удобства установки* – попытка проверить и устранить недостатки установочных процедур системы.

15. *Тестирование удобства эксплуатации* – попытка решить проблемы эргономике по взаимодействию пользователя с системой.

Анализ видов комплексного тестирования показал, что они не сводятся к проверке отдельных функций системы, а проверяют работоспособность КИС в целом. Чаще всего они пишутся в форме сценариев, в которых указываются действия, совершаемые во время выполнения теста. Все тестовые сценарии должны быть методологически и систематически проработаны. При тестировании должны быть использованы исходные данные и последовательности команд, которые в документации пользователя явно не

рекомендуются или объявляются запрещенными. На рисунке 1.8 изображен сценарий комплексного тестирования.

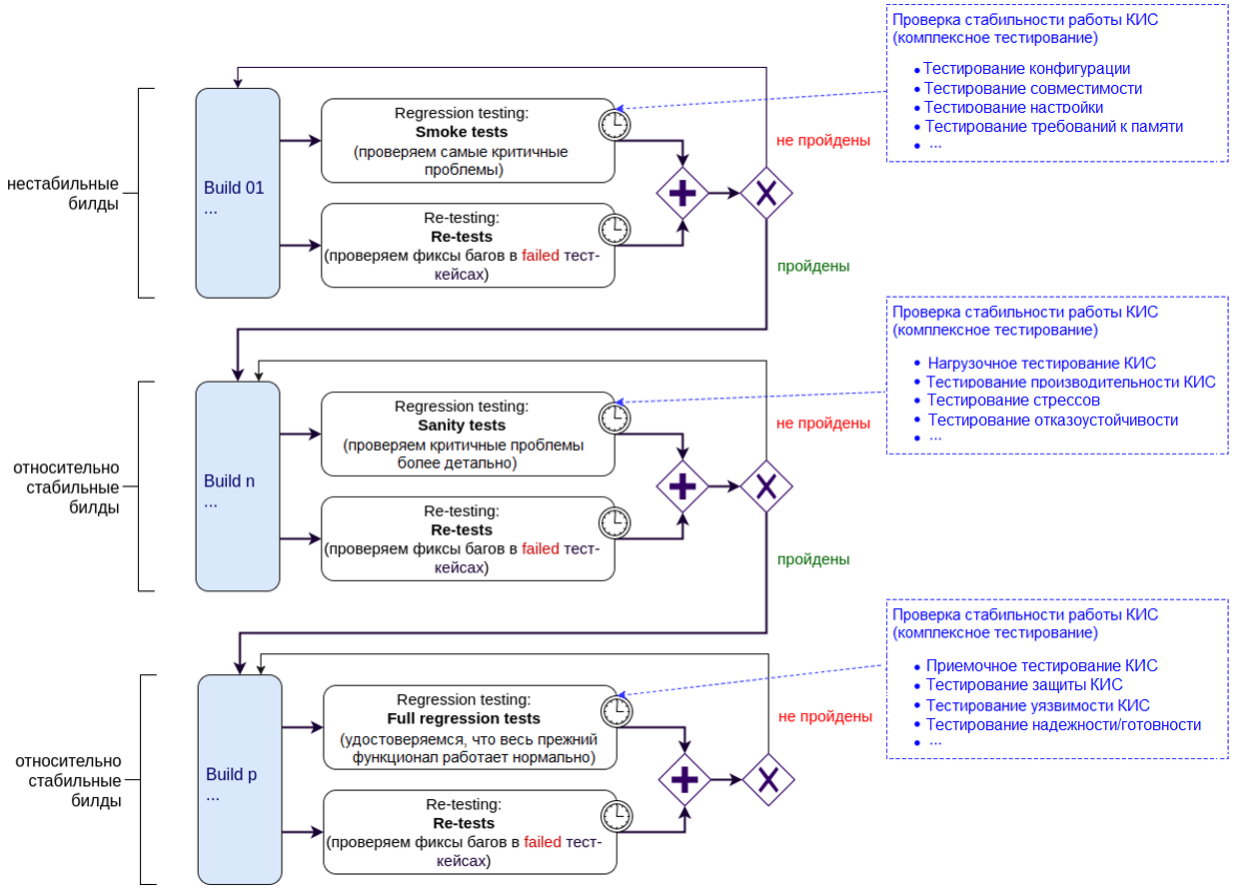


Рисунок 1.8– Сценарий комплексного тестирования КИС

В ходе анализа теоретических основ комплексного тестирования корпоративных систем и анализа его основных видов, был сделан вывод о том, что комплексное тестирование КИС не сводится к проверке ее функция, а ориентирован на сценку качества и производительности. Для проведения комплексного тестирования должен быть разработан сценарий, который должен предусмотреть все возможные варианты, обеспечивающие нестабильную работу КИС.

ГЛАВА 2 АНАЛИЗ МЕТОДОВ ПРОВЕДЕНИЯ КОМПЛЕКСНОГО ТЕСТИРОВАНИЯ КОРПОРАТИВНОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ

2.1 Основные критерии эффективности тестирования корпоративной информационной системы

Под тестированием понимают процесс исполнения программ на конечном множестве входных данных X, получения отклика Y и его сравнения с эталонным множеством выходных значений $Y_{эт}$, с целью выявления ошибок и дефектов в ИС [40].

Пара $(x, y_{эт}): x \in X, y_{эт} \in Y_{эт}$ называется тестовым случаем (тест-кейс), а все тестовые случаи, сгруппированные по определенному признаку, именуются тестовым комплектом.

Решение о наличии ошибки в программном обеспечении (ПС) КИС принимается либо при несовпадении результатов на одном из тестовых случаев:

$$\exists i,\ y^i_{эт} \in Y_{эт},\ y_i \in Y:\ \ y^i_{эт} \neq y_i \tag{2.1}$$

либо, если отличаются законы распределения выходных данных (рис. 2.1).

В обоих случаях считается, что тестирование прошло успешно, поскольку выявлена как минимум одна ошибка.

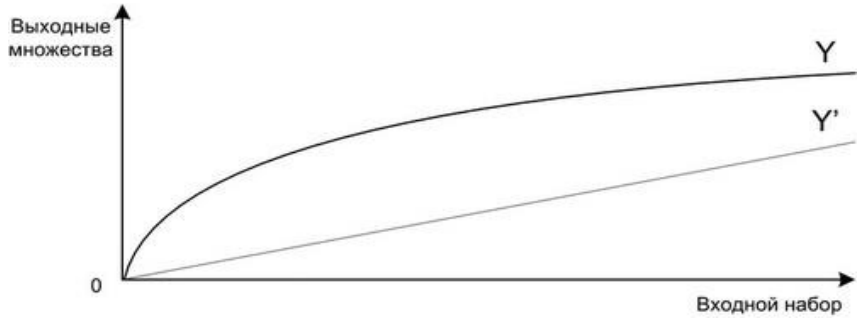


Рисунок 2.1 – Графики зависимостей результирующих и эталонных выходных данных от входного комплекта

Интенсивность обнаружения ошибок на единицу затрат и надежность тесно связаны со временем тестирования и, поэтому, с гарантией качества продукта (рисунок 2.2, блок А). Движение к убыванию числа оставшихся ошибок

или к качеству ИС приводит к использованию разных способов тестирования в процессе создания ПО. На рисунке 2.2 (блок В) приведен затратный компонент тестирования в зависимости от совершенствования применяемого инструментария и методов тестирования.

На практике известны последующие способы тестирования, упорядоченные по связанным с их применением затратам [43]:

- статический способ тестирования;
- модульный способ тестирование;
- интеграционный способ тестирования;
- системный способ тестирования;
- тестирование реального окружения и в реальном времени.

Зависимость эффективности внедрения перечисленных способов или их возможности к обнаружению соответственных классов ошибок (блок С) сопоставлена на рисунке с затратами. График указывает, что со временем, по мере обнаружения наиболее серьезных ошибок и недостатков, эффективность низкозатратных методов падает вместе с количеством обнаруживаемых ошибок.

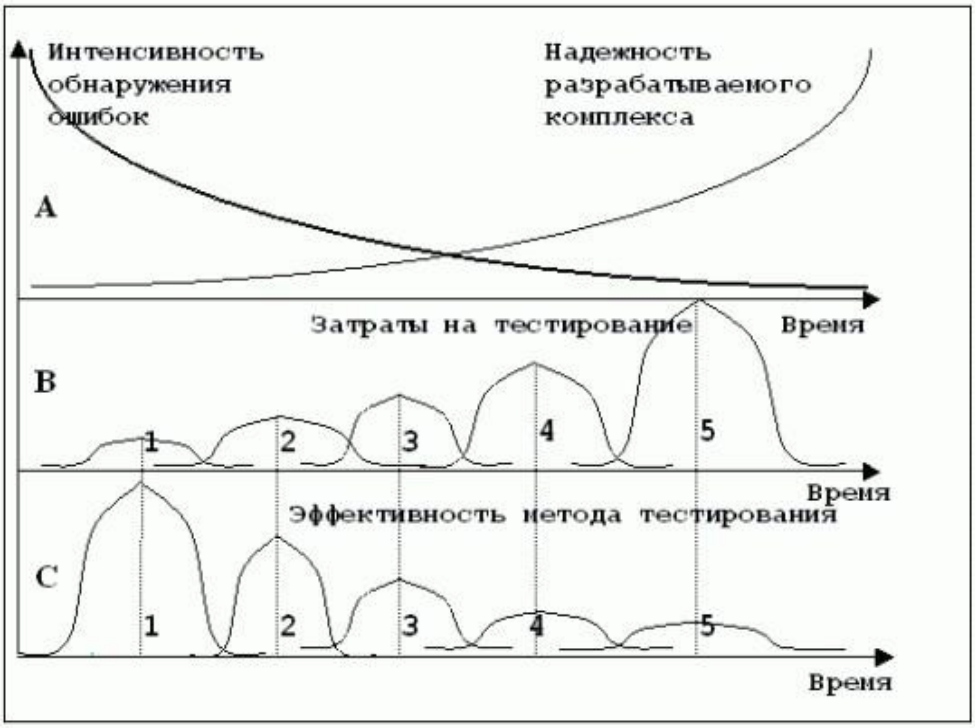


Рисунок 2.2 – Анализ эффективности критериев тестирования в процессе создания программного продукта

Отсюда следует, что все способы тестирования не только имеют право на существование, но и имеют свою нишу, где они хорошо обнаруживают ошибки, тогда как вне ниши их эффективность падает. Можно выделить требования к идеальному критерию тестирования:

- достаточный,
- полный,
- надежный,
- легко проверяемый.

При этом, существуют следующие классы критериев [27]:

1. Структурные критерии, использующие модель программы в виде «белого ящика», что предполагает знание исходного текста программы или спецификации программы в виде потокового графа управления. К ним относятся:
   - тестирование команд (критерий C0);
   - тестирование ветвей (критерий C1);
   - тестирование путей (критерий C2).
2. Функциональные критерии, использующие модель «черного ящика», что предполагает формулировку в описании требований к программному изделию и обеспечение контроля степени выполнения требований заказчика в программном продукте. К ним относятся:
   - тестирование пунктов спецификации;
   - тестирование классов входных данных;
   - тестирование правил;
   - тестирование классов выходных данных;
   - тестирование функций;
   - комбинированные критерии для программ и спецификаций.
3. Критерии стохастического тестирования, которые формулируются в терминах проверки наличия заданных свойств у тестируемого приложения, когда набор детерминированных тестов (X, Y) имеет громадную мощность. Критерии стохастического тестирования:

- статистические методы окончания тестирования – стохастические методы принятия решений о совпадении гипотез о распределении случайных величин (метод Стьюдента (St), метод Хи-квадрат (x^2) и т.п.);

- метод оценки скорости выявления ошибок – основан на модели скорости выявления ошибок, согласно которой тестирование прекращается, если оцененный интервал времени между текущей ошибкой и следующей слишком велик для фазы тестирования приложения.

4. Мутационные критерии, ориентированные на проверку свойств программного изделия на основе подхода Монте-Карло, позволяющего на основе мелких ошибок оценить общее число ошибок, оставшихся в программе.

Для разработки тестов следует нужно выбрать методы, реализующие критерии функционального тестирования, а также рассмотреть модель предметной области разработки.

2.2 Методы комплексного тестирования корпоративной информационной системы

Комплексное тестирование КИС предполагает применение методов тестирования программных продуктов и анализа предметной области.

1. Выделим среди методов тестирования: методы функциональных диаграмм и попарного тестирования [32, 36], а для анализа предметной области разработаем онтологическую модель.

Функциональная диаграмма представляет собой формальный язык, на который транслируется спецификация программы, написанная на естественном языке. Построение тестов этим методом осуществляется в несколько этапов:

- входные данные предметной области разбиваются на классы эквивалентности;

- по спецификации определяются причины и следствия, при этом под причиной понимают отдельное входное значение или класс эквивалентности входных данных, а следствие – это выходное значение или преобразование

программы (действие, которое входное условие оказывает на состояние программы);

- причины и следствия преобразуется в булевский граф – это и есть функциональная диаграмма;
- диаграмма добавляется примечаниями, задающими ограничения и описывающими комбинации причин и (или) следствий, которые являются невозможными из-за синтаксических или внешних ограничений;
- диаграмма преобразуется в таблицу решений с ограниченными входами, каждый столбец которой соответствует тесту;
- столбцы таблицы решений преобразуются в тесты.

При разработке тестов часто приходится анализировать работу системы с большим числом параметров (например, работа сайта в различных браузерах). Одним из подходов оптимизации количества тестов является использование ортогональных массивов.

Ортогональный массив – это таблица Lm(kn), где m – число строк, n – число столбцов, которое соответствует числу входных параметров, k – количество вариантов значений элементов таблицы, и обладающая следующими свойствами:

- любые два столбца таблицы содержат все комбинации значений этих столбцов;
- если какая-либо пара значений двух столбцов встречается несколько раз, то все возможные парные комбинации значений этих столбцов должны встретиться столько же раз.

Для тестирования с использованием ортогональных массивов следует выполнить следующие шаги:

- задать комбинации переменных для входных данных;
- определить значения, которые могут принимать переменные;
- построить ортогональный массив, который имеет столбец для каждой переменной;

- поставить в соответствие каждому тестовому случаю комбинацию значений переменных, расположенных в строке построенного массива.

На протяжении многих лет, был разработан целый ряд комбинаторных стратегий для того, чтобы помочь тестировщикам выбрать такое подмножества входных комбинаций, которое позволило бы максимально увеличить вероятность выявления дефектов: выборочное тестирование, «каждый-выбор» (each-choice) и «основание выбора» (base choice), антирандомизация (antirandom), стратегия тестирование t-способами (t-wise testing strategies) и другие. Парное тестирование (pairwise testing) является наиболее видным среди них. На рисунке 2.3 показана зависимость увеличения числа исчерпывающих и парных тестов от количества тестовых уровней (возможного количества значений параметров).

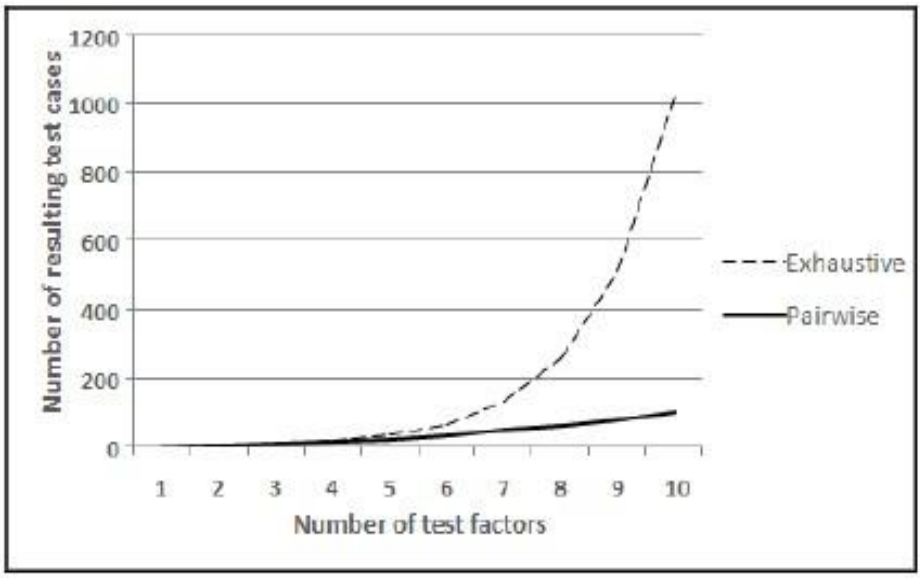


Рисунок 2.3 – Зависимость увеличения числа исчерпывающих и парных тестов от количества тестовых уровней

Парное тестирование - метод проектирования черного ящика, в котором тестовые случаи предназначены для выполнения всех возможных дискретных комбинаций каждой пары входных параметров.

Выходные данные программного приложения зависят от многих факторов, например, от входных параметров, переменных состояния и конфигурации среды. Парное тестирование использует: анализ граничных значений и

распределение эквивалентности, которые предполагают, что для каждой пары входных параметров системы должны существовать все возможные дискретные комбинации этих параметров.

Формально стратегия парного тестирования определяется следующим образом: дан набор из N независимых испытаний факторов f1,f2, ..., fN, где каждый фактор fi имеет Li возможных уровней fi = {li,1, ..., li, Li}, и набор тестов R. Каждый тест в R содержит N тест-уровней, по одному для каждого тест-фактора fi, и, в совокупности, все тесты в R охватывают все возможные пары уровней тест-факторов (относящихся к разным параметрам). Другими словами, для каждой пары уровней факторов li,p и lj,q, где $1 \leq p \leq Li$, $1 \leq q \leq Lj$, и $i \neq j$ – существует по крайней мере один тест в R, который содержит li,p и lj,q.

Для анализа предметной области разработки тестов применяется онтологический подход [35]. Под онтологией понимается упорядоченная тройка вида:

$$O = \langle C, R, F \rangle, \tag{2.2}$$

где

C – конечное множество концептов (понятий, терминов) предметной области, которую представляет онтология O;

R – конечное множество отношений между концептами заданной предметной области;

F – конечное множество функций интерпретации (аксиоматизации), заданных на концептах и/или отношениях онтологий O.

Тестирования следует осуществлять с точки зрения пользователя, что предполагает полное понимание того, для чего система будет применяться.

На рисунках 2.4, 2.5 представлены онтологические модели тестового случая (классическое представление и реализация в программной среде).

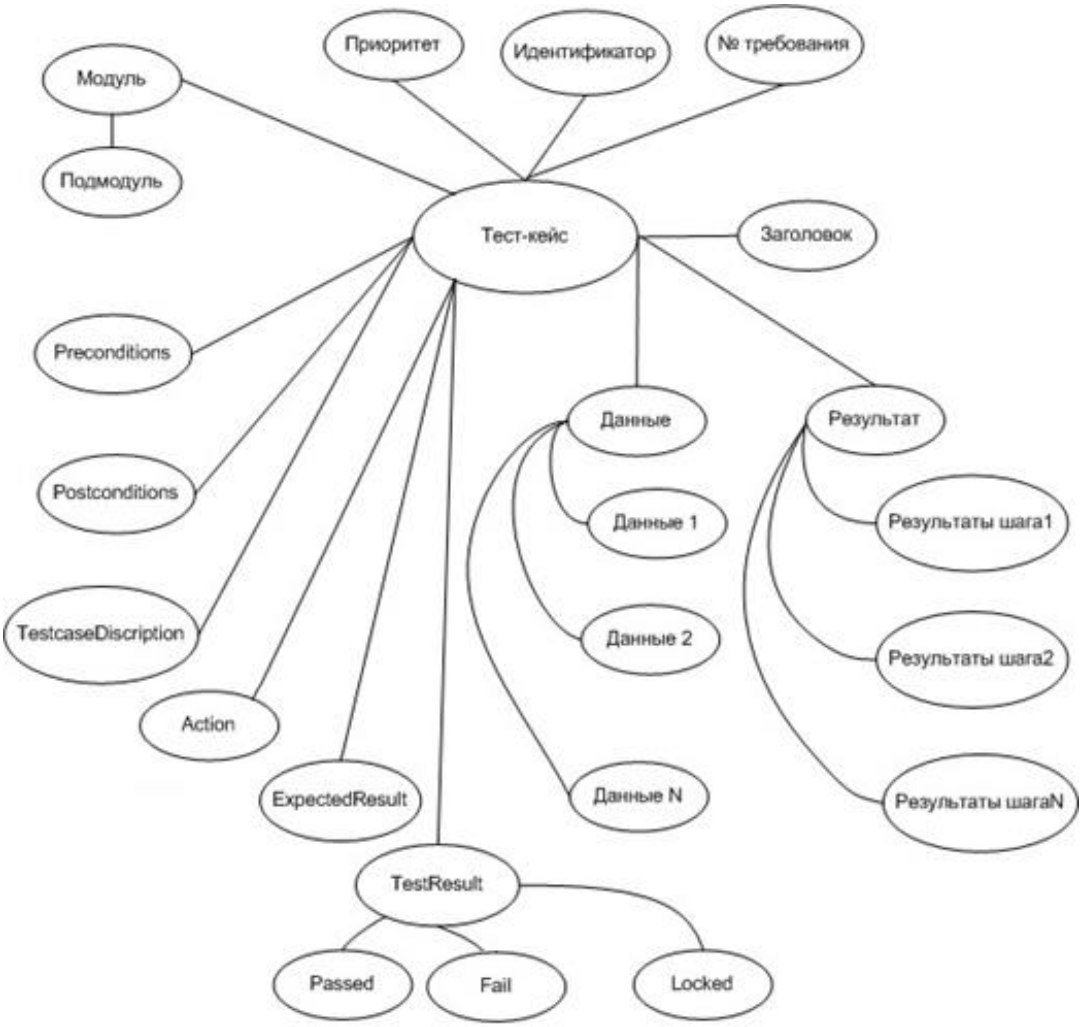


Рисунок 2.4 – Онтологическая модель тестового случая

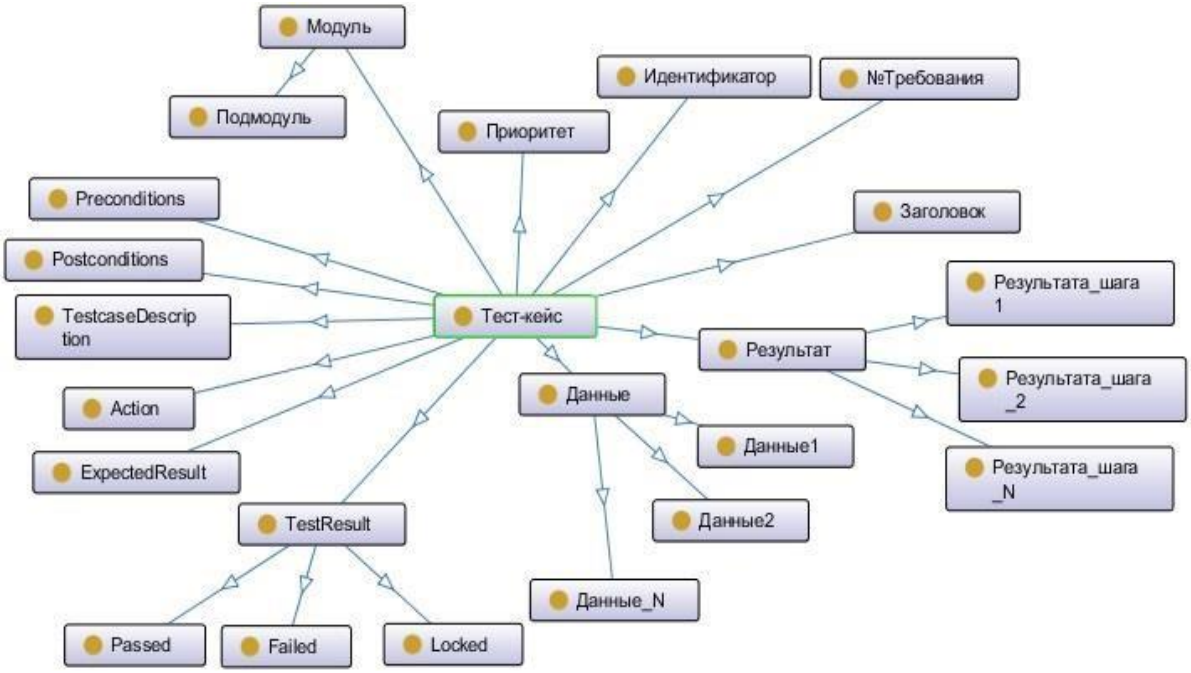


Рисунок 2.5– Онтологическая модель тестового случая в Protege

2. В связи с тем, что современному классу корпоративных информационных систем присущ модульный принцип их построения, т.е. их интеграция, выделим два подхода к комбинированию модулей [38]: пошаговое и монолитное тестирование.

Первый подход - *монолитный метод,* или *метод «большого удара»,* применяемый при тестировании и сборке программы.

Второй подход - *пошаговый метод* тестирования или сборки, предполагающий, что модули тестируются не изолированно друг от друга, а подключаются поочередно для выполнения теста к набору уже ранее оттестированных модулей до тех пор, пока к набору оттестированных модулей не будет подключен последний модуль.

Анализ выделенных методов модульного тестирования КИС показал:

1. Монолитное тестирование требует больших затрат труда.

2. При монолитном тестировании меньше расход машинного времени.

3. Монолитный метод предоставляет большие возможности для параллельной организации работы на начальной фазе тестирования (тестирования всех модулей одновременно).

4. При пошаговом тестировании раньше обнаруживаются ошибки в интерфейсах между модулями, поскольку раньше начинается сборка программы.

5. При пошаговом тестировании легче отладка программ.

6. Результаты пошагового тестирования более совершенны и дают более точный анализ в определении дефектов КИС.

Все это позволяет нам сделать вывод, что *пошаговое тестирование является предпочтительным* при разработке и апробации КИС.

3. Интеграционное тестирование направлено на проверку взаимодействия между частями (модулями) программного обеспечения КИС. Убедившись в преимуществах пошагового тестирования перед монолитным, исследуем две возможные стратегии тестирования: *нисходящее* и *восходящее* [6, 8, 13, 36].

Существуют два основных подхода к проведению интеграции: восходящая интеграция (рис. 2.6); нисходящая интеграция (рис. 2.7).

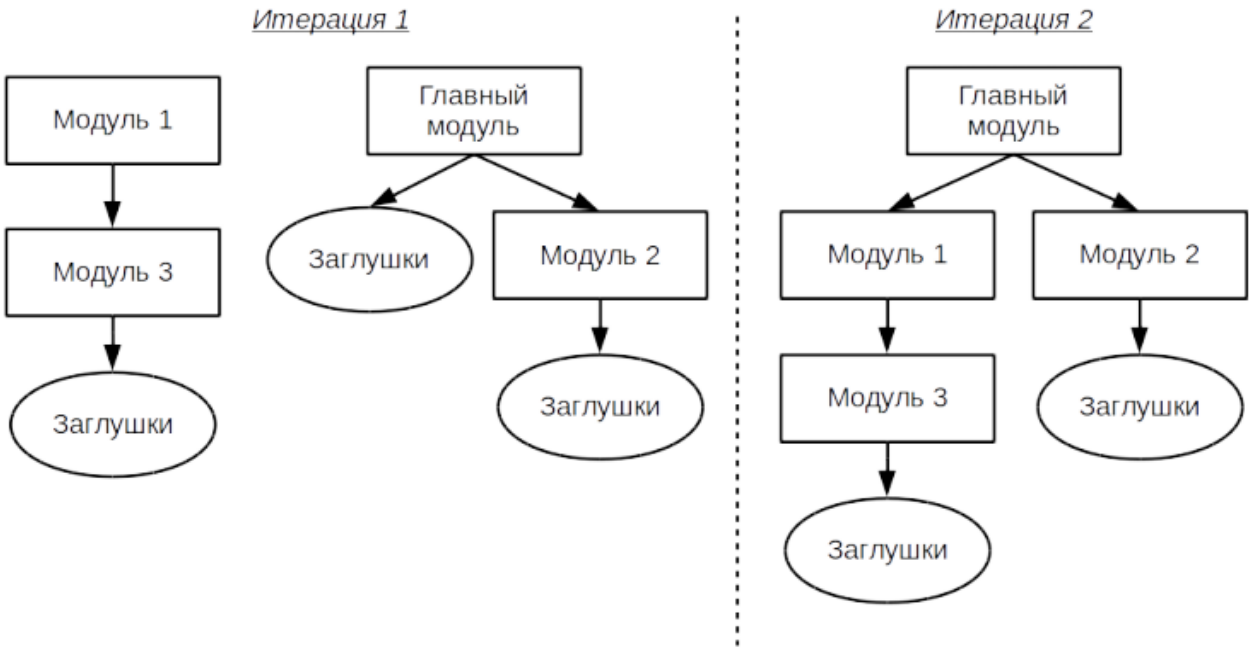


Рисунок 2.6 – Схема восходящей интеграции

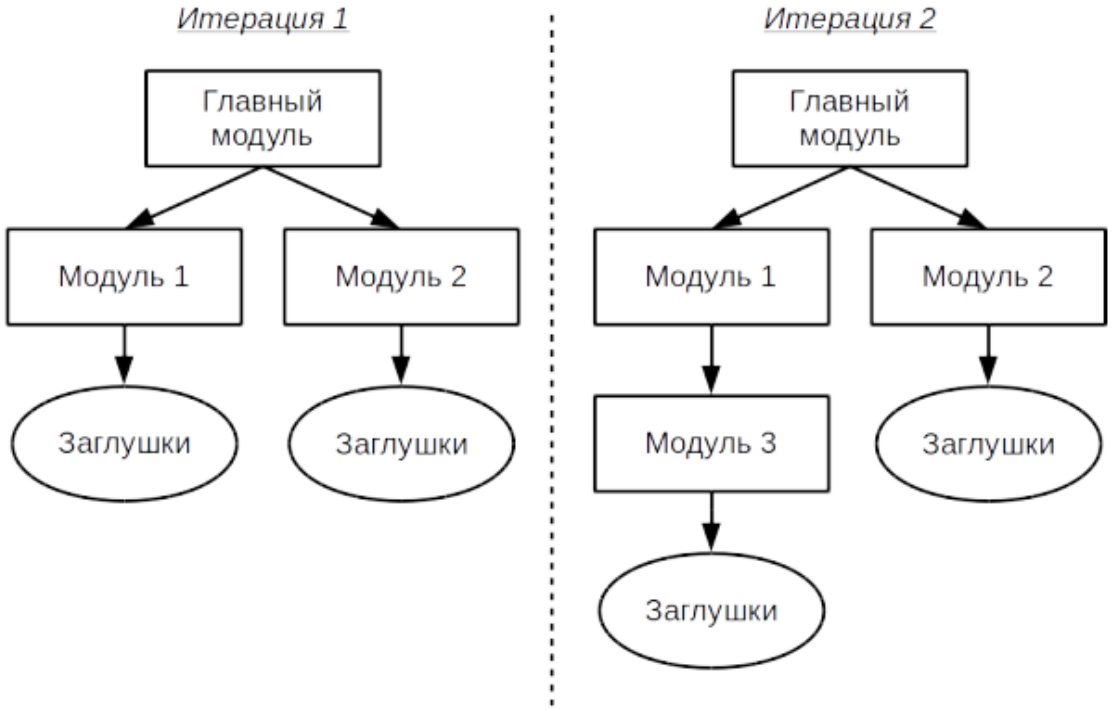


Рисунок 2.7 – Схема нисходящей интеграции

Преимущества и недостатки представленных подходов отражены в таблице 2.1.

Таблица 2.1 - Сравнение способов интеграции

	Восходящее	**Нисходящее**
Преимущества	Возможность ранней проверки корректности низкоуровневого поведения Не требуется написание заглушек Просто определить требования ко входам/выходам модулей	Возможность ранней проверки корректности высокоуровневого поведения Модули могут добавляться по одному, независимо друг от друга Не требуется разработка множества драйверов Можно разрабатывать систему как в глубину, так и в ширину
Недостатки	Отложенная проверка высокоуровневого поведения Требуется разработка драйверов При замене драйвера на модуль высокого уровня может произойти «миниБольшой Взрыв» числа найденных ошибок	Отложенная проверка низкоуровневого поведения Требуется разработка «заглушек» Крайне сложно корректно сформулировать требования ко входам/выходам частичной системы

4. Для того чтобы процесс тестирования КИС имел оправданную с экономической точки зрения трудоемкость, необходимо заранее выработать ряд стратегий. Наиболее распространенные:

- тестирование методом «черного» ящика - тестирование, управляемое данными (data-driven testing) или тестирование, управляемое входом и выходом (input/output-driven testing);

- тестирование методом «белого» ящика - тестирование, управляемое логикой программы (logic- driving testing).

В таблице 2.2 представлена разница между тестированием черного ящика и тестированием белого ящика.

Таблица 2.2 – Сравнительный анализ методов комплексного тестирования

Тестирование черного ящика	**Тестирование белого ящика**
Тестирование черного ящика – это метод тестирования программного обеспечения, который используется для тестирования программного обеспечения, не зная внутренней структуры кода или программы.	Тестирование белого ящика – это метод тестирования программного обеспечения, при котором внутренняя структура известна тестировщику, который собирается тестировать программное обеспечение.
Проводится тестерами.	Проводится разработчиками программного обеспечения.
Знания о внедрении не требуются для проведения тестирования черного ящика.	Знания о внедрении требуется для проведения тестирования белого ящика.
Знания по программированию не требуются.	Знания по программированию необходимы
Тестирование применимо на более высоких уровнях тестирования, таких как системное тестирование, приемочное тестирование.	Тестирование применимо на более низких уровнях тестирования, таких как модульное тестирование, интеграционное тестирование.
Означает функциональное тестирование или внешнее тестирование.	Означает структурные испытания или внутренние испытания.
Концентрируется на функциональности тестируемой системы.	Концентрируется на тестировании программного кода тестируемой системы, такой как структура кода, ветви, условия, циклы и т.д.
Основная цель – проверить, какие функции выполняет тестируемая система.	Основная цель – проверить, как работает система.
Может быть начато на основании документов с техническими требованиями.	Может быть начато на основании документов рабочего проекта.
Функциональное тестирование, тестирование поведения, тестирование закрытого бокса выполняется в рамках тестирования «черного ящика», поэтому знания в области программирования не требуются.	Структурное тестирование, логическое тестирование, тестирование контуров, циклическое тестирование, тестирование покрытия кода, тестирование Open Box выполняется в рамках тестирования «белого ящика», поэтому необходимо обязательно знать о знаниях программирования.

5. Для комплексного тестирования КИС также применяются следующие методы [37]:

1. Альфа-тестирование – это ручное тестирование потенциальными пользователями, заказчиками или независимой командой тестирования на стенде разработки. Альфа-тестирование часто используется как форма внутреннего приемочного тестирования перед проведением бета-тестирования.

Альфа-тестирование позволяет фильтровать, уточнять и передавать разработчикам поступающие дефекты с подробным описанием, что значительно сокращает время, а также позволяет сокращать трудозатраты разработчиков на поиск причины дефекта и его исправление.

В рамках проведения альфа-тестирования решаются следующие задачи:

- подготовка расписания тестирования;
- организация участников тестирования;
- отбор и уточнение поступающих замечаний;
- регистрация дефектов.

2. Бета-тестирование проводится после альфа-тестирования и может использоваться как приемочное тестирование внешними пользователями. Бета-версия системы передается группе пользователей вне команды разработки, чтобы снизить количество дефектов. Иногда версия передается нескольким командам, чтобы получить обратную связь от как можно большего количества будущих пользователей.

Ключевые преимущества:

- получение отзывов и пожеланий от потенциальных пользователей КИС;
- повышение качества проведенного тестирования в заданные сроки и ликвидация проблем, связанных с тестовой средой.

Таким образом, использование в совокупности рассмотренных методов способствует более качественному проведению комплектного тестирования КИС.

2.3 Комплексное тестирование корпоративной информационной системы на ее производительность

Современные корпоративные информационные системы часто вынуждены работать с максимальной нагрузкой. Проявляющиеся при этом сбои и ошибки ведут к убыткам, которые часто невозможно оценить. Тестирование производительности позволяет минимизировать эти риски и решить целый ряд связанных с ними проблем.

Тестирование производительности позволяет оценивать и планировать производительность КИС, а также управлять ею в условиях плановой, повышенной и пиковой нагрузки. Нагрузочное тестирование выполняется для оценки поведения системы в нормальных условиях и при прогнозируемом пике нагрузки. Стрессовое тестирование предполагает тщательную проверку в экстремальных условиях для оценки стабильности КИС.

2.3.1 Тестирование производительности КИС

Тестирование производительности (Performance Testing) – это тестирование, которое выполняется для определения того, как компоненты системы работают в конкретной ситуации [32]. Основная цель тестирования производительности включает установление эталонного поведения системы. Тестирование производительности не направлено на поиск дефектов в приложении. Успешный тест производительности должен спроектировать большинство проблем производительности, которые могут быть связаны с базой данных, сетью, программным обеспечением, оборудованием и т.д.

Тестирование производительности проводится с целью предоставления заинтересованным сторонам информации об их приложении, касающейся скорости, стабильности и масштабируемости. Тестирование производительности определяет, соответствует ли программное обеспечение ИС требованиям скорости, масштабируемости и стабильности при ожидаемых рабочих нагрузках. Тестирование производительности проводится, чтобы убедиться, что приложение работает достаточно быстро, чтобы удерживать

внимание и интерес пользователя. Тесты производительности могут указывать на потенциальное влияние изменений на производительность, особенно после внесения изменений в технологию, реализацию или конфигурацию.

Ниже приведен общий процесс тестирования производительности [27]:

1. *Определите свою среду тестирования* – ознакомьтесь с деталями аппаратного, программного обеспечения и сетевых конфигураций, использованных во время тестирования.

2. *Определите критерии приемлемости производительности* – сформулируйте цели и ограничения пропускной способности, времени отклика и распределения ресурсов, критерии успеха проекта вне этих целей и ограничений.

3. *Планируйте и проектируйте тесты производительности* – определите, как использование может варьироваться среди конечных пользователей, и определите ключевые сценарии для тестирования во всех возможных случаях использования.

4. *Настройка среды тестирования* – подготовьте среду тестирования перед выполнением, организуйте инструменты и другие ресурсы.

5. *Реализовать тестовый дизайн* – создайте тесты производительности в соответствии с тестовым дизайном.

6. *Запустить тесты* – выполните и оцените результаты тесты.

7. *Анализируйте, настраивайте и повторно тестируйте* – объединяйте, анализируйте и делитесь результатами испытаний; выполните точную настройку и проверьте снова, чтобы увидеть, есть ли улучшение или снижение производительности.

Создаваемые сценарии тестирования производительности должны быть близки к реальным условиям. Технология тестирования производительности должна поддерживать сценарии, которые не только увеличивают количество пользователей, но и имитируют их поведение. Типичным поведением может быть, например, посещение пользователем домашней страницы, вход в систему, переход по ссылке на статью, добавление товара в корзину и совершение покупки.

В следующем фрагменте кода показано описание простого моделирования для Gatling на Scala (рис. 2.8).

```
import io.getling.core.Predef/_
import io.getling.core.structure.ScenarioBuilder
import io.getling.http.Predef/_
import io.getling.http.protocol.HttpProtocolBuildder
import scala.concurrent.duration._

class CarCreationSimulation extends Simulation {
    val httpConf: HttpProtocolBuildder = http
    .baseURL(http://test.car-manufacture.example.com/car-
manufacture/resources)
        .acceptHeader("*/*")

    val scn: ScenarioBuilder = scenario("create_car")
        .exec(http('request_1")
            .get("/cars"))
        .exec(http('request_1")
            .post("/cars"))
            .body(StringBody("""{"id": "X123A234", "color": "RED",
                        "engine": "DIESEL"}""")).asJSON
            .check(header("Location").saveAs("locationHeader")))
        .exec(http('request_1")
            .get("${locationHeader}"))

    Pause(1 second)

□etup(
    scn.inject(rampUsersperSec(10).to(20).during(10 second))
        ).protocols(httpConf)
        .constantPauses
}
```

Рисунок 2.8 – Фрагмент описания обработки клиентских запросов

Сценарий create_car включает в себя три клиентских запроса, которые читают список всех автомобилей, создают автомобиль и подключаются к

созданному ресурсу. Сценарии настраивают несколько виртуальных пользователей. Количество пользователей начинается с 10 и увеличивается до 20 в течение десяти секунд. Тестирование производительности позволяет определить максимальную интенсивность операций, при которой система удовлетворяет требованиям ко времени отклика.

На рынке доступно множество инструментов для тестирования производительности:

- NeoLoad – это платформа для тестирования производительности, разработанная для DevOps, которая легко интегрируется в существующий конвейер непрерывной доставки. С NeoLoad команды тестируют в 10 раз быстрее, чем с традиционными инструментами, чтобы соответствовать новому уровню требований на протяжении всего жизненного цикла разработки программного обеспечения Agile – от компонентов до полных нагрузочных тестов в масштабе всей системы;
- LoadView Testing – платформа для тестирования инфраструктуры в любом масштабе. LoadView предлагает нагрузочное тестирование на основе облачных вычислений по требованию.
- HP LoadRunner – это самый популярный на сегодняшний день инструмент для тестирования производительности. Этот инструмент способен моделировать сотни тысяч пользователей, подвергая приложения реальной нагрузке, чтобы определить их поведение при ожидаемой нагрузке. Loadrunner имеет виртуальный генератор пользователей, который имитирует действия живых пользователей.
- Jmeter – один из ведущих инструментов, используемых для нагрузочного тестирования веб-серверов и серверов приложений.

Тестирование производительности рекомендуется проводить для новой версии ПО, планируемого к внедрению. Оно позволяет выявить и предотвратить отказы КИС перед внедрением в промышленную эксплуатацию; определить максимальное количество одновременно работающих пользователей в системе и

максимальное количество одновременно выполняемых операций без потери качества обслуживания.

2.3.2 Нагрузочное тестирование КИС

Нагрузочное тестирование (load testing) – данный тип тестирования позволяет оценить поведение системы при возрастающей нагрузке. Целью нагрузочного тестирования является определение максимальной нагрузки, которую может выдержать система [28].

В роли нагрузки может выступать количество пользователей, а также количество операций на сервере.

Производительность при этом определяется следующими факторами:

- скоростью работы программного обеспечения;
- скоростью работы аппаратного обеспечения;
- скоростью работы сети.

Во время тестирования могут осуществляться следующие операции, позволяющие более точно измерить производительность и определить «узкое место» системы:

- измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций;
- определение количества пользователей, одновременно работающих с приложением;
- определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций).

Пример нагрузочного теста показан на рисунке 2.9. Этот тест проводится для определения количества пользователей, с которыми может работать система. В этом тесте 100 пользователей добавляются через каждые 30 секунд, пока нагрузка не достигнет 1000 пользователей. Каждый шаг занимает 30 секунд, и Jmeter ждет 30 секунд, прежде чем начать следующий шаг. Как только нагрузка достигнет 1000 потоков, все они будут продолжать работать в течение 300 секунд (5 минут) вместе, а затем, наконец, останавливают 10 потоков каждые 3 секунды.

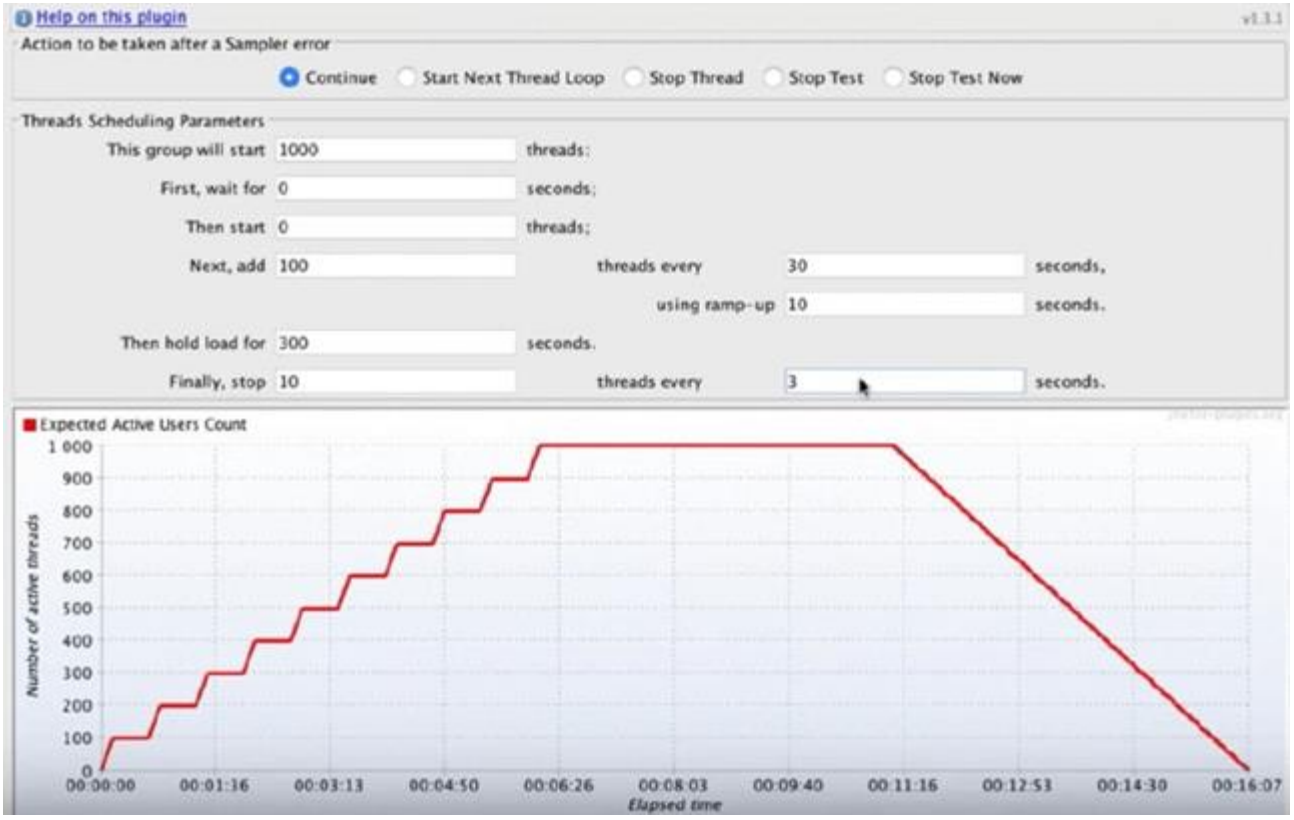


Рисунок 2.9 – Пример выполнения нагрузочного тестирования

Другой пример нагрузочного теста показан на рис. 2.10, где изображен тест на пики при внезапном увеличении количества пользователей до 7000.

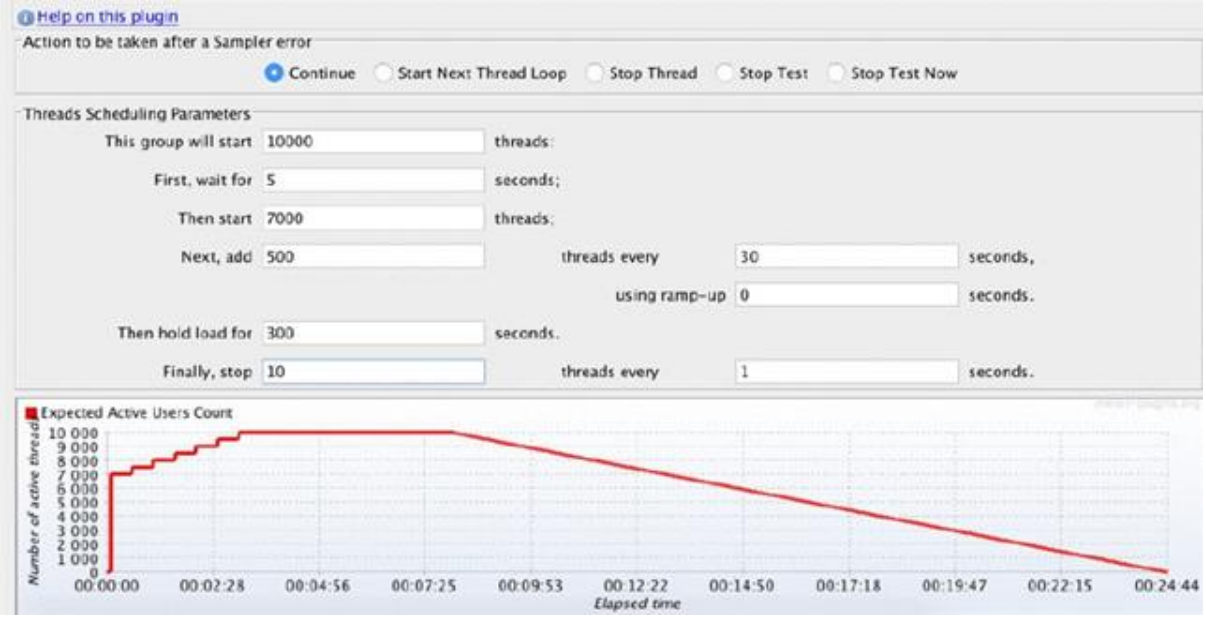


Рисунок 2.10 – Пример выполнения нагрузочного тестирования

Нагрузочное тестирование – исследование способности приложения сохранять заданные показатели качества при нагрузке в допустимых пределах и некотором превышении этих пределов (определение «запаса прочности»).

2.3.3 Стресс-тестирование КИС

При стресс-тестировании выполняются различные действия по перегрузке существующих ресурсов избыточными заданиями в попытке сломать систему [44]. Отрицательное тестирование, которое включает удаление компонентов из системы, также проводится в рамках стресс-тестирования. Это тестирование**,** также известное как усталостное тестирование, должно отражать стабильность приложения путем его тестирования за пределами пропускной способности.

Цель стресс-тестирования состоит в том, чтобы установить отказ системы и отследить, как система корректно восстанавливается. Задача здесь состоит в том, чтобы настроить контролируемую среду перед запуском теста, чтобы вы могли точно фиксировать поведение системы при самых непредсказуемых сценариях.

Цель стресс-тестирования состоит в том, чтобы проанализировать отчеты после аварии, чтобы определить поведение приложения после сбоя. Самая большая проблема состоит в том, чтобы гарантировать, что система не ставит под угрозу безопасность конфиденциальных данных после сбоя. При успешном стресс-тестировании система вернется в нормальное состояние вместе со всеми ее компонентами даже после самого страшного сбоя.

Подводя итог, в таблице 2.3 представлены основные различия между этими тремя типами тестирования.

Таблица 2.3 - Сравнительный анализ методов тестирования производительности

	Тестирование производительности	**Нагрузочное тестирование**	**Стресс-тестирование**
Домен	Суперсет нагрузочного и стресс-тестирования	Подмножество тестирования производительности.	Подмножество тестирования производительности.
Объем	Очень широкая сфера применения. Включает нагрузочное тестирование, стресс-тестирование, тестирование емкости, объемное тестирование, тестирование на выносливость, тестирование пиков, тестирование на масштабируемость и тестирование надежности и т.д.	Более узкий охват по сравнению с тестированием производительности. Включает объемные испытания и испытания на выносливость.	Более узкий охват по сравнению с тестированием производительности. Включает тестирование на впитывание и тестирование шипов.
Основная цель	Чтобы установить ориентир и стандарты для приложения.	Чтобы определить верхний предел системы, установите SLA приложения и посмотрите, как система обрабатывает тома с большой нагрузкой.	Определить, как система ведет себя при интенсивных нагрузках и как восстанавливается после сбоя. В основном, чтобы подготовить ваше приложение к неожиданному скачку трафика.
Предел нагрузки	Оба – ниже и выше порога перерыва.	До порога разрыва	Выше порога разрыва

	Тестирование производительности	**Нагрузочное тестирование**	**Стресс-тестирование**
Атрибуты изучены	Использование ресурсов, надежность, масштабируемость, использование ресурсов, время отклика, пропускная способность, скорость и т.д.	Пиковая производительность, пропускная способность сервера, время отклика при различных уровнях нагрузки **(**ниже порога перерыва**),** адекватность рабочей среды, количество пользовательских приложений, требования к балансировке нагрузки и т.д.	Стабильность за пределами пропускной способности, время отклика **(**выше порога разрыва) и т.д.
Проблемы, выявленные с помощью этого типа тестировани я	Все ошибки производительности, включая раздувание во время выполнения, возможности оптимизации, проблемы, связанные со скоростью, задержкой, пропускной способностью и т.д.	Проблемы с балансировкой нагрузки, проблемы с пропускной способностью, проблемы с пропускной способностью системы, плохое время отклика, проблемы с пропускной способностью и т.д.	Пробелы в безопасности с перегрузкой, проблемы с повреждением данных в ситуации перегрузки, медлительность, утечки памяти и т.д..

В рамках тестирования производительности собирается статистика использования системы, на основе которой составляется профиль нагрузки. После этого вычисляется начальная точка и размер шага для увеличения интенсивности выполнения операций.

В ходе выполнения тестирования определяется максимальная производительность системы: наибольшая интенсивность выполнения операций с требуемым качеством обслуживания (SLA); пиковая производительность системы, при которой происходит ухудшение показателей качества обслуживания операций (время выполнения, отказы).

2.4 Методы и сценарий приемочного тестирования корпоративных информационных систем

Приемочное тестирование – это комплексное тестирование, необходимое для определения уровня готовности системы к последующей эксплуатации. Тестирование проводится на основании набора тестовых сценариев, покрывающих основные бизнес-операции системы [41].

Ключевые преимущества:

- позволяет обнаружить системные нарушения;
- позволяет обнаружить дефекты, связанные с удобством и простотой использования.
- привлечение опытных компетентных специалистов позволяет грамотно, качественно и в заданные сроки провести процесс приемки тестирования.

Основные этапы приемочного тестирования:

1. *Подготовка* – включает разработку программы и методики испытаний и подготовку приемочных тестов.

2. *Проведение* – сопровождение клиента во время проведения приемочных тестов (заведение дефектов, отслеживание корректности и скорости выполнения тестирования).

3. *Отчет* – клиенту предоставляется подробный отчет с перечнем ошибок, которые нужно устранить перед запуском системы в эксплуатацию.

Направления приемочного тестирования:

- операционное тестирование – проверка системы на способность выполнять свою роль в среде эксплуатации согласно бизнес-модели;

- альфа-тестирование – проверка независимой командой тестирования;
- пользовательское тестирование – проверка пригодности системы для внедрения конечными пользователями;
- бета-тестирование – тестирование внешними пользователями, потенциальными клиентами.

Сценарий приемочного тестирования должен содержать четко сформулированные разделы, представленные на рисунке 2.11.

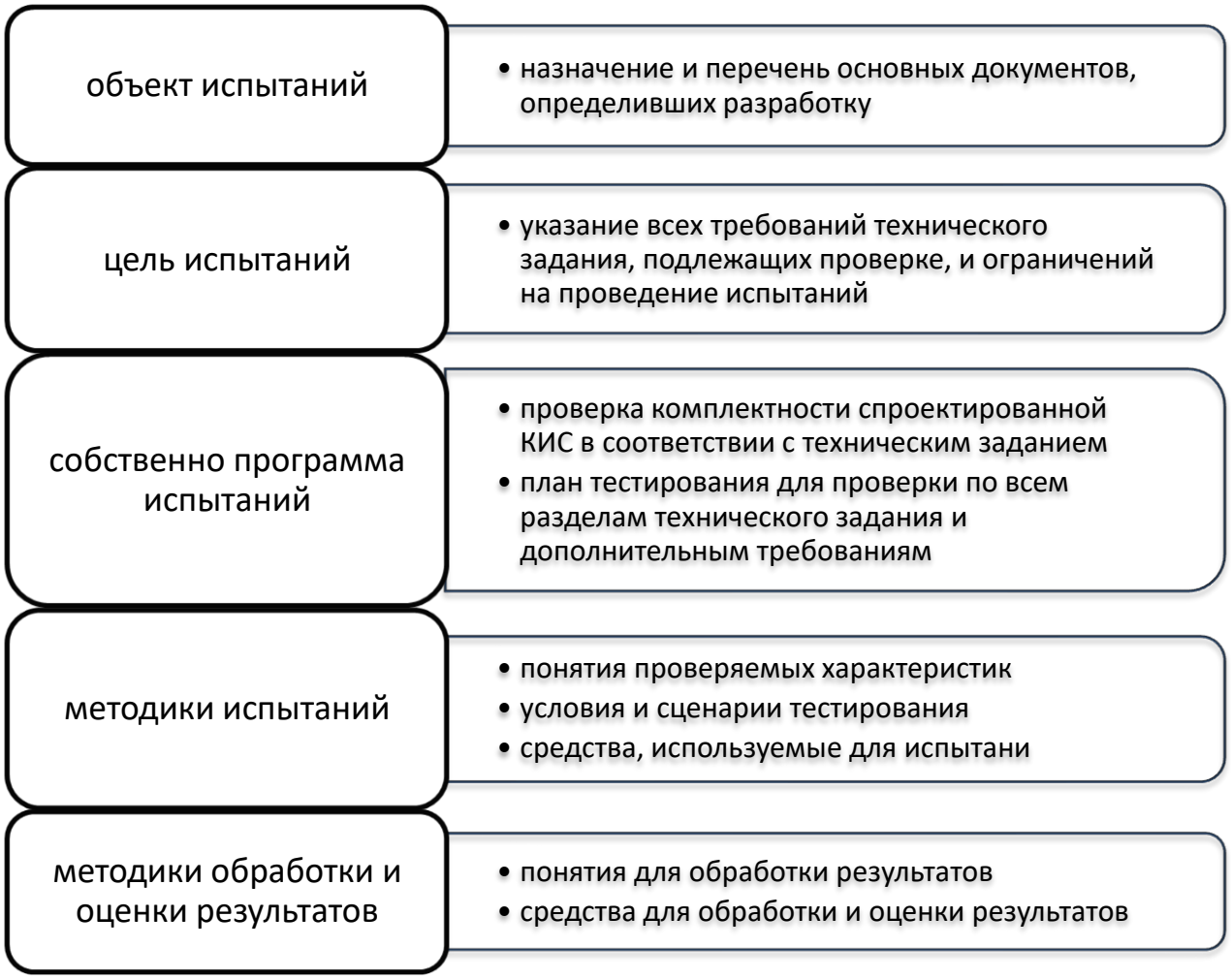


Рисунок 2.11 – Сценарий приемочного тестирования

Наиболее полным и разносторонним приемочным испытаниям должна подвергаться первая базовая версия КИС. Комплексное тестирование программного обеспечения КИС гарантируют проверку выполнения всех требований технического задания. Предложенный

сценарий проведения приемочного тестирования позволяет фиксировать условия неправильной работы программ и характер проявления дефектов.

При завершающих испытаниях основное внимание, кроме проверок функциональной пригодности, должно сосредоточиваться на подготовке стрессовых тестов, тестировании в режимах предельного использования ресурсов, надежности функционирования КИС. Для этого необходимо проводить испытания в два последовательных этапа – Альфа- и Бета-тестирование [12, 26]. Объем тестов и длительность обоих этапов тестирования будут зависеть от сложности комплекса программного обеспечения КИС и интенсивности потока изменений.

ГЛАВА 3 АПРОБАЦИЯ МЕТОДОВ КОМПЛЕКСНОГО ТЕСТИРОВАНИЯ КОРПОРАТИВНОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ

3.1 Нагрузочное тестирование, основанное на моделях

Рассмотрим подход, основанный на моделях, применительно к нагрузочному тестированию КИС, в рамках которого осуществляется тестирование прикладного программного обеспечения (ПО) в IT-среде функционирования, включающей СУБД, базы данных, системное ПО, развернутые на оборудовании комплекса технических средств ИС [29, 30, 31].

При нагрузочном тестировании КИС в процессе их эксплуатации главной целью является не проверка правильности функционирования прикладных программ (предполагается, что функциональное тестирование ПО проведено), а оценка эксплуатационных характеристик информационной системы, в составе которой функционирует прикладное ПО. Обычно это так называемые «Показатели назначения», или нефункциональные требования, то есть пропускная способность/производительность ИС, реактивность при решении функциональных задач и ряд других.

При формализации предметной области (для класса систем) формируются исходные данные [30]:

- информация о выбранном виде нагрузочного тестирования (оценочное, аналитическое, настроечное, регрессионное);
- информация об измеряемых характеристиках и показателях производительности;
- информация о структуре системы с точки зрения вариантов подачи нагрузки и способов измерений;
- информация о планируемой загрузке в структурированном виде.

Представим следующие модели, с помощью которых при постановке задачи нагрузочного эксперимента осуществляется выбор необходимых характеристик, показателей и измеряемых величин, адекватно характеризующих процесс функционирования тестируемой ИС (рис. 3.1):

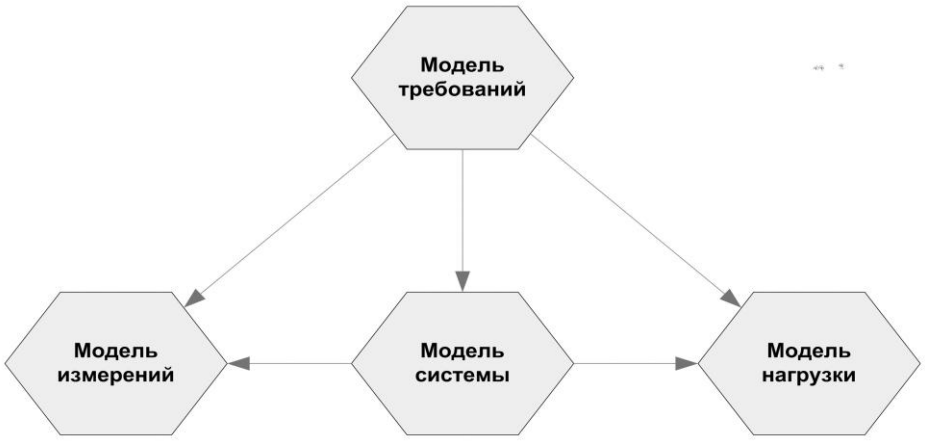


Рисунок 3.1 – Взаимосвязь моделей

1. Модель требований – характеризует тип тестируемой системы, состав нефункциональных требований (бизнес-правила и технические требования) тестируемой информационной системы [31].

Модель требований может быть представлена в следующем виде:

$$R = B \cup T \tag{3.1}$$

где

R – множество требований к системе;

B – множество бизнес-правил, связанных с технологическими процессами, корпоративными регламентами, политиками, стандартами, законодательными актами, внутрикорпоративными инициативами, учетными практиками, алгоритмами вычислений и т.д.;

T – множество технических требований, устанавливающих технические свойства, которыми должна обладать система, например, характеристики производительности, надежности и доступности.

Модель требований представляет собой вербальное описание нефункциональных требований, сгруппированных по типам и объектам, к которым они применимы. Эта модель содержит также критерии оценки получаемых в результате нагрузочного тестирования характеристик и расчетных показателей.

2. Модель системы – описывает структуру системы как сеть систем массового обслуживания (включая состав элементов типа «ресурс») [31].

Модель системы может быть представлена в следующем виде:

$$\Sigma = \{U(p)\},\{S\},K_S,K_U \quad (3.2)$$

где

$\{U(p)\}$ – множество устройств объекта тестирования с характеристиками производительности p;

$\{S\}$ – множество программных комплексов (компонент, сервисов);

K_S – матрица связей между программными комплексами, строками которой являются источники, столбцами – приемники, а в ячейках указывается наличие связи между ними;

K_U – матрица связей программных комплексов с устройствами, характеризующая количество выделенных устройством ресурсов для программного комплекса. Строками этой матрицы являются программные комплексы, столбцами – вычислительные комплексы. Элементами матрицы являются векторы выделенных ресурсов.

Матрицы K_S и K_U для представленной схемы выглядят следующим образом:

$$K_S = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \quad K_U = \begin{pmatrix} p_1 & 0 & 0 & 0 \\ 0 & p_2 & 0 & 0 \\ 0 & 0 & p_3 & 0 \\ 0 & 0 & 0 & p_4 \\ 0 & 0 & 0 & p_4 \end{pmatrix}$$

где p_i – вектор характеристик производительности, характеризующий количество выделенных на вычислительном комплексе ресурсов для заданного программного комплекса.

Модель системы представляет собой описание объекта тестирования, в которое входят программные и технические средства, их связи и выделяемые ресурсы (память, процессор и т.п.). На рисунке 3.2 представлен пример модели системы в виде графической схемы взаимодействия программных и вычислительных комплексов.

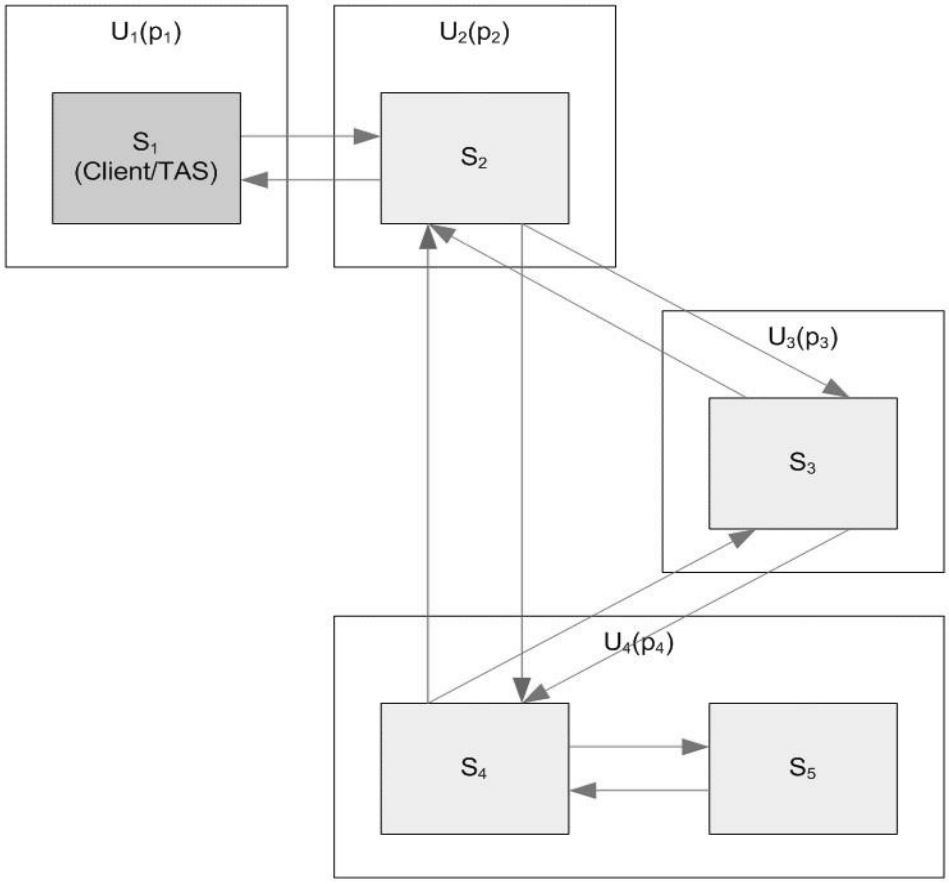


Рисунок 3.2 – Пример модели системы в виде графической схемы

3. Модель нагрузки – представляет собой описание количества и типов требований обслуживания к системе, закон распределения требований обслуживания во времени эксперимента, правила поступления требований обслуживания в систему, точки входа требований обслуживания в систему (логический уровень) [29].

Модель нагрузки может быть представлена в следующем виде:

$$L = \{F,M,I\} \tag{3.3}$$

где

F – множество функций, характеризующих распределения нагрузки, вводимой в систему;

M – многомерная матрица, размерностями которой могут являться виды нагрузки: источники потоков нагрузки, наименования и типы потоков, а элементами – их количественные характеристики;

I – множество интерфейсов для ввода нагрузки (в виде ссылки на модель системы).

Поток нагрузки структурируется по его динамическим (*F*) и статическим (*M*) свойствам.

Модель нагрузки представляет собой описание тестовых данных и правил их поступления в систему. Фактически по модели нагрузки формируются исходные данные для генерации нагрузки инструментальными средствами.

4. Модель измерений – определяет состав собираемых характеристик, показателей и величин, интерфейс ввода требований в систему, метод их сбора и алгоритмы преобразования, а также критерии оценки полученных результатов [28].

Модель измерений предназначена для унификации описания:

- способов получения измеряемых величин при нагрузочном тестировании ИС;
- принципиальных возможностей постановки процесса измерений;
- типовых способов оценки показателей и характеристик;
- общих свойств инструментальных средств для анализа возможностей их использования для автоматизации измерений.

Модель измерений может быть представлена в следующем виде:

$$\Delta = \{|U|\}, \tau, \mu, R, \omega \quad (3.4)$$

где

$\{|U|\}$ – перечень измеряемых величин для каждого типа устройств информационной системы или ее части; τ – периодичность и скважность измерений; μ – множество расчетных показателей и их взаимосвязь с измеряемыми величинами;

R – типовые правила и алгоритмы получения расчетных показателей;

ω – типовые критерии оценки полученных результатов.

Для планирования нагрузочного эксперимента ключевую роль играет выбор перечня измеряемых характеристик производительности (табл. 3.1), так

как на основе полученных значений этих характеристик делаются выводы о результатах эксперимента.

Таблица 3.1 – Виды характеристик производительности

Характеристики производительности	Расчетные показатели
Реактивность	Среднее время отклика
	Среднее время ожидания
	Среднее время обслуживания
Продуктивность	Пропускная способность
	Выработка
Использование	Утилизация ресурса
	Относительная пропускная способность

Реактивность важна для КИС, эксплуатационными характеристиками ИС могут быть время отклика (реакции) системы на запрос пользователя или время ожидания решения задачи (например, может решаться задача минимизации времени ожидания выполнения бизнес-транзакции).

Характеристики реактивности определяются как время между предъявлением системе входных данных и появлением соответствующих выходных данных. Реактивность оценивают или рассчитывают на основе следующих показателей производительности:

- среднее время отклика;
- среднее время обслуживания;
- среднее время ожидания.

Время отклика рассчитывается как период между началом транзакции и завершением выполнения последнего этапа транзакции.

Время отклика (*Tr*) обычно складывается из времени обслуживания (время, за которое производится обработка) (*Ts*) и времени ожидания (время ожидания ресурса) (*Tw*): $Tr = Ts + Tw$.

Показатели реактивности зависят от типа системы, структуры ее точек входа и выхода.

Для КИС важна их интегральная пропускная способность, оцениваемая как количество бизнес-транзакций, обрабатываемых системой в единицу времени (продуктивность) [38].

При оценке продуктивности, как правило, используют непосредственно измеренные или расчетные значения следующих показателей производительности:

- выработка (*V*);
- пропускная способность (или абсолютная пропускная способность) ©.

Выработка (*V*) определяется как количество завершенных транзакций за определенный период времени:

$$V_i = \frac{N_i}{T} \tag{3.5}$$

где

$i = 1, \ldots, n$ - порядковый номер периода времени;

N_i - количество завершенных транзакций;

T - период времени.

Пропускная способность (*C*) - максимально возможное количество завершенных транзакций за единицу времени:

$$C = \max(V_1, \ldots, V_n) \tag{3.6}$$

Характеристики использования предназначены для того, чтобы описать, в какой степени используются ресурсы тестируемой системы при заданной нагрузке.

К использованию относятся следующие показатели производительности:

- утилизация ресурса (коэффициент использования ресурса);
- относительная пропускная способность.

Утилизация ресурса показывает, какую часть времени ресурс используется для обслуживания транзакций. Общая формула:

$$U = \frac{\sum_{i=1}^{n} t_{Si}}{T} \times 100\% \tag{3.7}$$

где

n - количество обслуженных транзакций;

t_{Si} - время обслуживания i-ой транзакции;

T - период обслуживания.

Характеристики использования ресурсов и их количество существенно зависят от используемого в системе оборудования и входящих в его состав ресурсов: процессоров, оперативной памяти, внешних носителей, каналов ввода-вывода и т.п.

Технология нагрузочного тестирования, основанная на моделях (рис. 3.3), состоит из следующих этапов [29]:

- определение целей тестирования - с помощью модели требований происходит определение правил формализации требований к эксплуатационным характеристикам системы и формируется модель требований;
- разработка программы и методики испытаний - модель нагрузки используется для определения возможной статической и динамической структуры и состава потока нагрузки;
- подготовка к тестированию - на основе модели нагрузки выполняется настройка средств планирования тестирования и генератора тестовых данных;
- подача нагрузки - средства автоматизации выполняют процедуры подачи нагрузки в точки входа, заданные для каждого типа нагрузки в модели системы;
- сбор данных - выполняется автоматизированный сбор значений измеряемых характеристик;
- интерпретация и анализ результатов - средствами автоматизации используются все четыре модели: модель требований для сопоставления результатов эксперимента с требованиями к системе; модель нагрузки и модель системы обеспечивают формирование отчета об условиях проведения эксперимента.

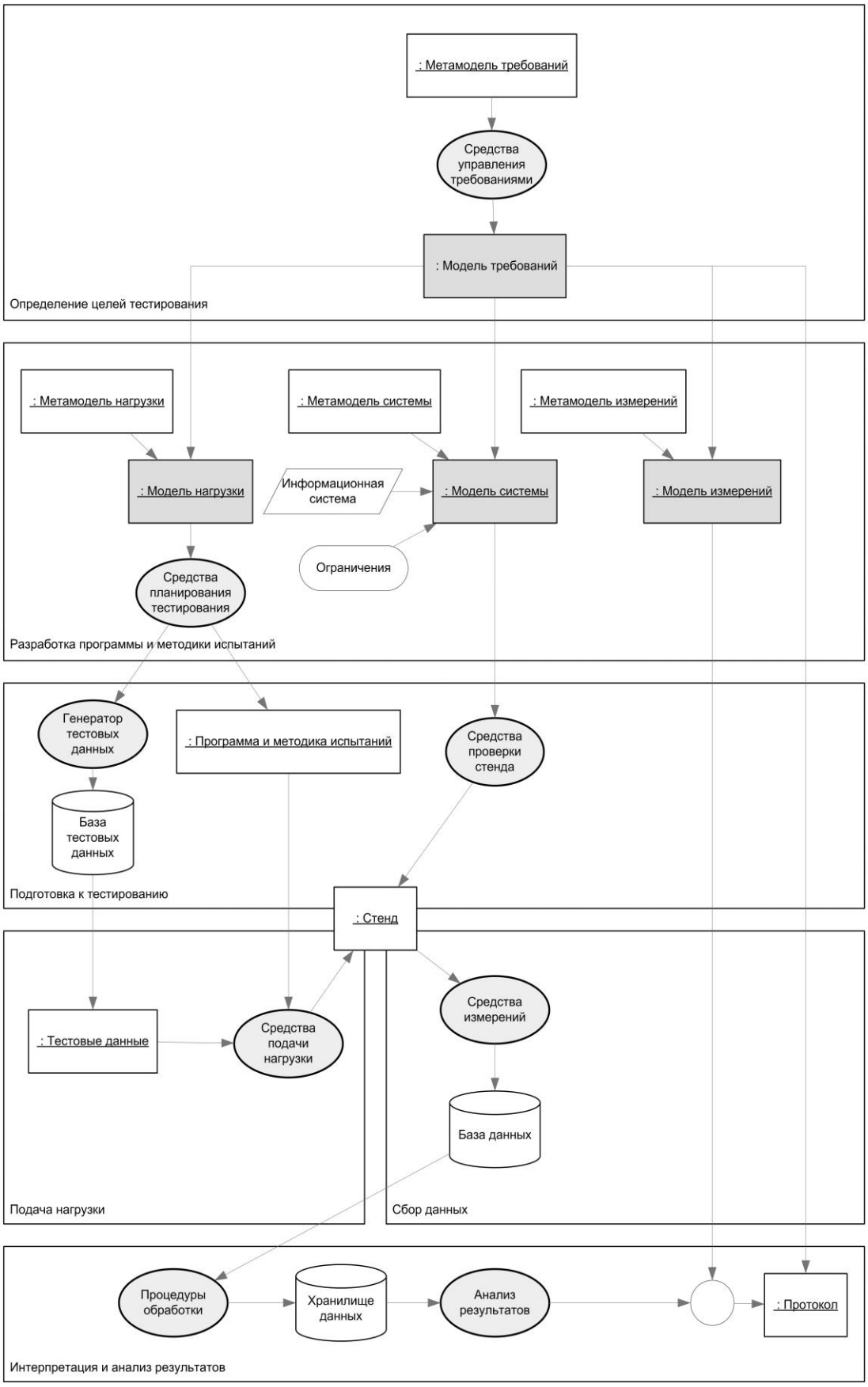


Рисунок 3.3 - Процесс нагрузочного тестирования с использованием моделей

Технология, основанная на использовании моделей, описывающих требования к проведению нагрузочного эксперимента и свойства объекта тестирования, охватывает все аспекты планирования, проведения нагрузочного эксперимента и анализа его результатов.

Использование моделей существенно (в несколько раз) снижает трудоемкость нагрузочного тестирования и обеспечивает возможность повторного использования подготовленного эксперимента, например, при контроле деградации информационной системы в ее жизненном цикле.

3.2 Сценарий нагрузочного тестирования с использованием различных инструментальных средств

Рассмотрим нагрузочное тестирование для оценки производительности простого HTTP-запроса GET из 20 потоков с 100 000 итераций.

Сторона сервера (тестируемое приложение): Процессор: 4x Xeon L5520 @ 2,27 ГГц; RAM: 8 ГБ; ОС: Microsoft Windows Server 2008 R2 x64; Сервер приложений: IIS 7.5.7600.16385.

Клиентская сторона (генератор нагрузки): Процессор: 4x Xeon L5520 @ 2,27 ГГц; RAM: 4 ГБ; ОС: Ubuntu Server 12.04 64-битная.

Для проведения тестирования будем использовать инструментальные средства, каждый инструмент будет отправлять запросы так быстро, как может:

1. *Grinder* - это бесплатная среда нагрузочного тестирования на основе Java, доступная по лицензии BSD в стиле open-source.

На рис. 3.4 представлен фрагмент проведения нагрузочного тестирования, выполненного с использованием Grinder.

```
root@perf-temp:/opt/grinder# cat perf-temp-0.log
             Tests         Errors        Mean Test    Test Time     TPS           Mean          Response      Response      Mean time to Mean time to Mean time to
                                         Time (ms)    Standard                   response      bytes per     errors        resolve host establish    first byte
                                                      Deviation                  length        second                      connection
                                                      (ms)

(Test 100    2000000       0             18.17        60.47         1073.24       0.00          0.00          0             -            -            0.00)          "Page 1"
Test 101     2000000       0             17.89        57.21         1073.24       2422.36       2599758.75    0             0.03         2.10         17.03          "GET /"

Totals       2000000       0             17.89        57.21         1073.24       2422.36       2599758.75    0             0.03         2.10         17.03
             (2000000)     (0)
```

Рисунок 3.4 - Нагрузочное тестирования с использованием Grinder

2. *Gatling Project* - это инструмент для тестирования производительности с открытым исходным кодом, который в основном разработан и поддерживается Stephane Landelle.

На рис. 3.5 представлен пример отчета Gatling для сценария загрузки.

Рисунок 3.5 – Пример отчета нагрузочного тестирования с Gatling

3. *Tsung* - инструмент для тестирования производительности с открытым исходным кодом. Tsung использует Erlang. Tsung не предоставляет графический интерфейс для разработки или выполнения тестов.

Основной вызов сценария tsung производит следующий вывод, представленный на рисунке 3.6. На рисунках 3.7 представлен статистический отчет нагрузочного тестирования, 3.8 – графический отчет.

```
blazemeter@perf-temp:~$ tsung
Usage: tsung <options> start|stop|debug|status
Options:
    -f <file>     set configuration file (default is ~/.tsung/tsung.xml)
                   (use - for standard input)
    -l <logdir>   set log directory where YYYYMMDD-HHMM dirs are created (default is ~/.tsung/log/)
    -i <id>       set controller id (default is empty)
    -r <command>  set remote connector (default is ssh)
    -s            enable erlang smp on client nodes
    -p <max>      set maximum erlang processes per vm (default is 250000)
    -m <file>     write monitoring output on this file (default is tsung.log)
                   (use - for standard output)
    -F            use long names (FQDN) for erlang nodes
    -w            warmup delay (default is 1 sec)
    -v            print version information and exit
    -6            use IPv6 for Tsung internal communications
    -x <tags>     list of requests tag to be excluded from the run (separated by comma)
    -h            display this help and exit
```

Рисунок 3.6 - Вызов сценария tsung

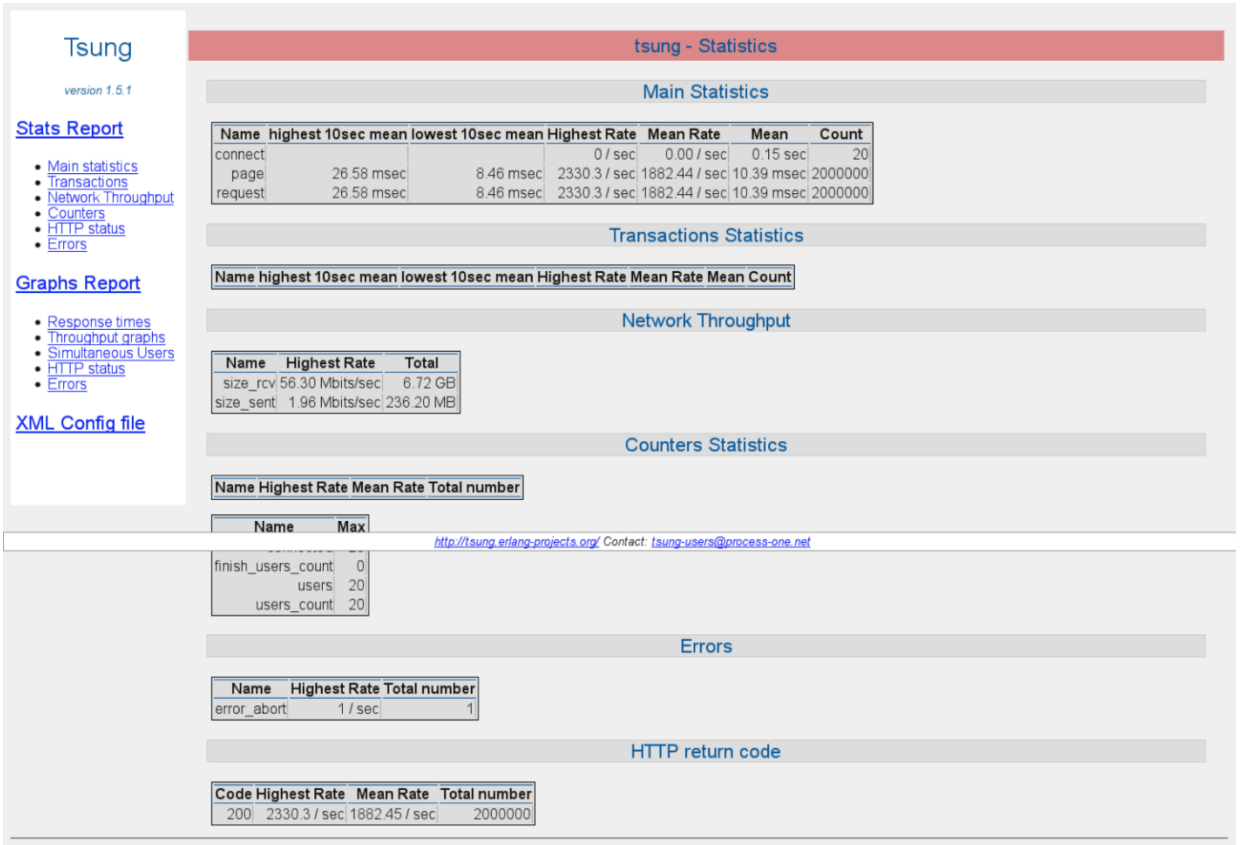


Рисунок 3.7 – Статистический отчет нагрузочного тестирования

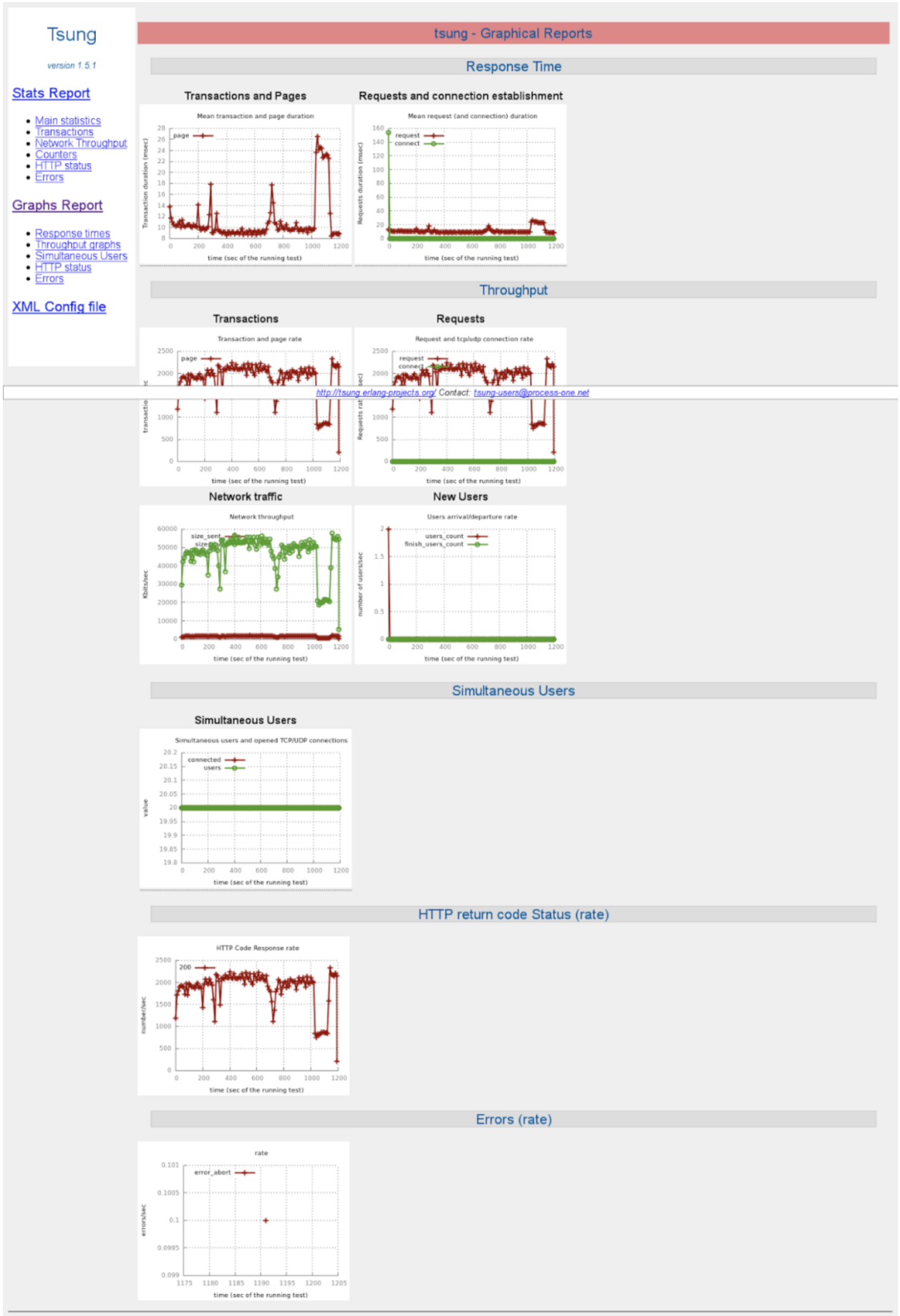


Рисунок 3.8 – Графический отчет нагрузочного тестирования

4. *Apache JMeter™* - имеет удобный графический интерфейс, что значительно облегчает разработку тестов и отладку. Приложение JMeter с агрегированным отчетом по сценарию нагрузки (рис. 3.9).

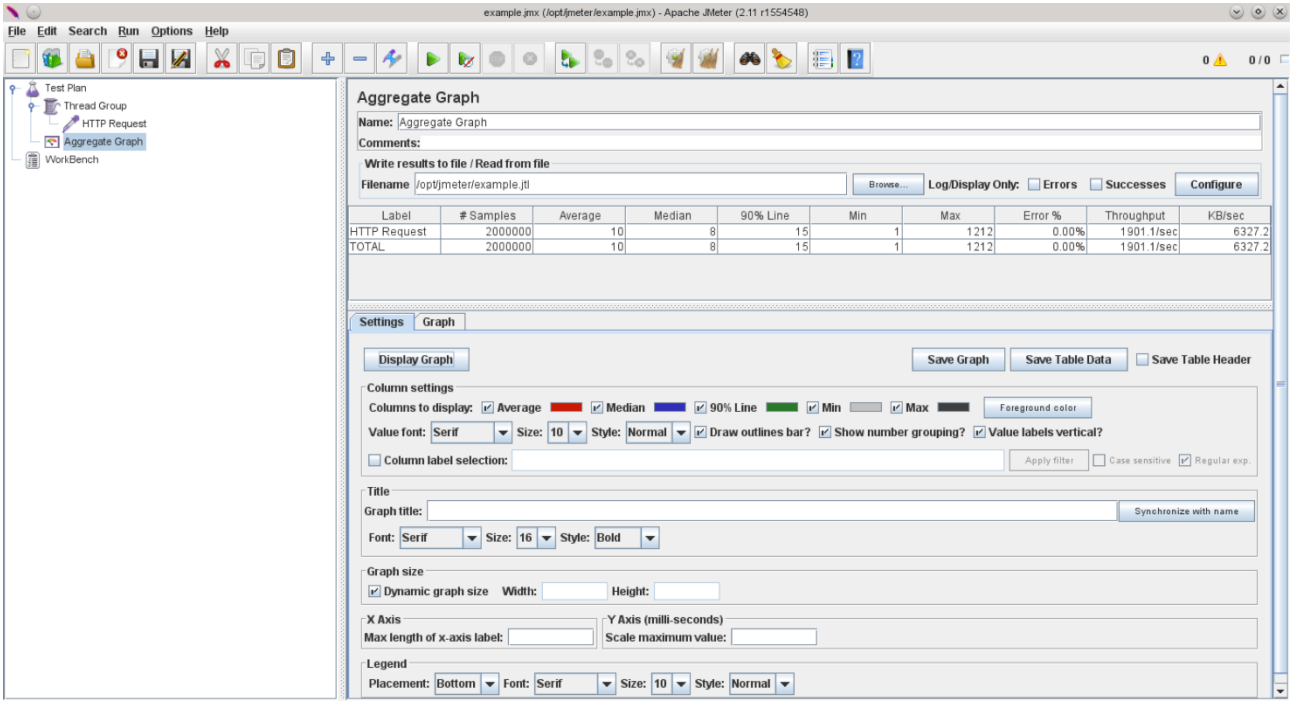


Рисунок 3.9 - Агрегированный отчет по сценарию нагрузки JMeter

5. *Locust* основанной на Python среде с открытым исходным кодом, которая позволяет писать скрипты производительности на чистом языке Python.

Пример базового тестового сценария с выводом представлен на рисунке 3.10.

```
from locust import HttpLocust, TaskSet, task
class SimpleLocustTest(TaskSet):
 @task
 def get_something(self):
 self.client.get("/")
class LocustTests(HttpLocust):
 task_set = SimpleLocustTest
```

```
→ JMeterVsLocust locust -f simple_locust_script.py --host=http://192.168.1.170:8080
[2017-11-18 14:46:38,458] Yuris-MBP-2.home/INFO/locust.main: Starting web monitor at *:8089
[2017-11-18 14:46:38,459] Yuris-MBP-2.home/INFO/locust.main: Starting Locust 0.8.1
```

Рисунок 3.10 – Пример тестового сценария Locust

После выполнения скрипта отображается подробная отчётность (рис. 3.11).

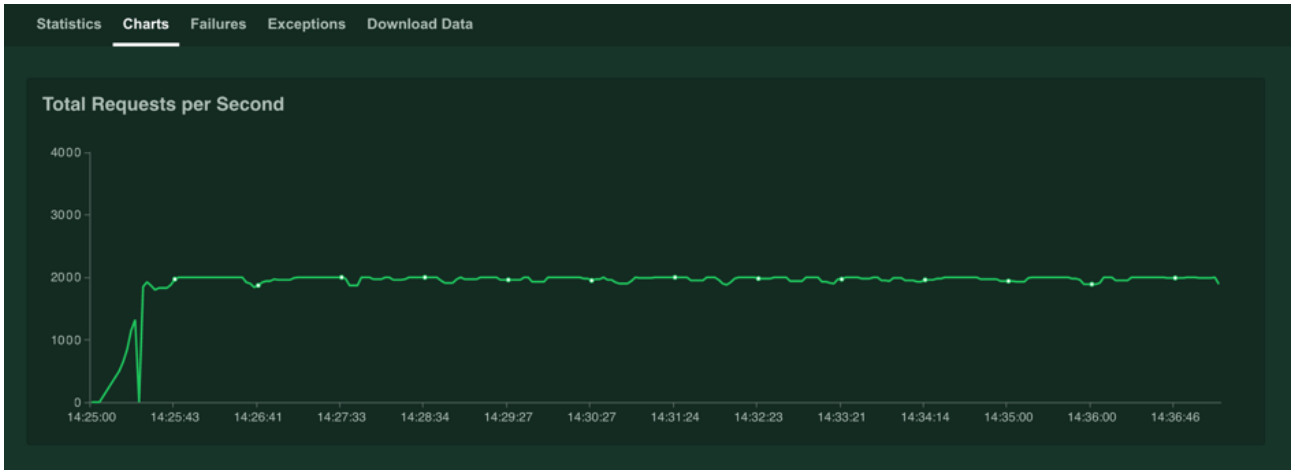


Рисунок 3.11 – Отчетность тестового сценария Locust

Сравним результаты нагрузочного теста этих инструментов со следующими показателями:

1. Среднее время отклика (мс).
2. Средняя пропускная способность (запросов / сек).
3. Общее время выполнения теста (минут).

На рисунках 3.12, 3.13 представлены диаграммы, отображающие средний ответ и общее время выполнения теста.

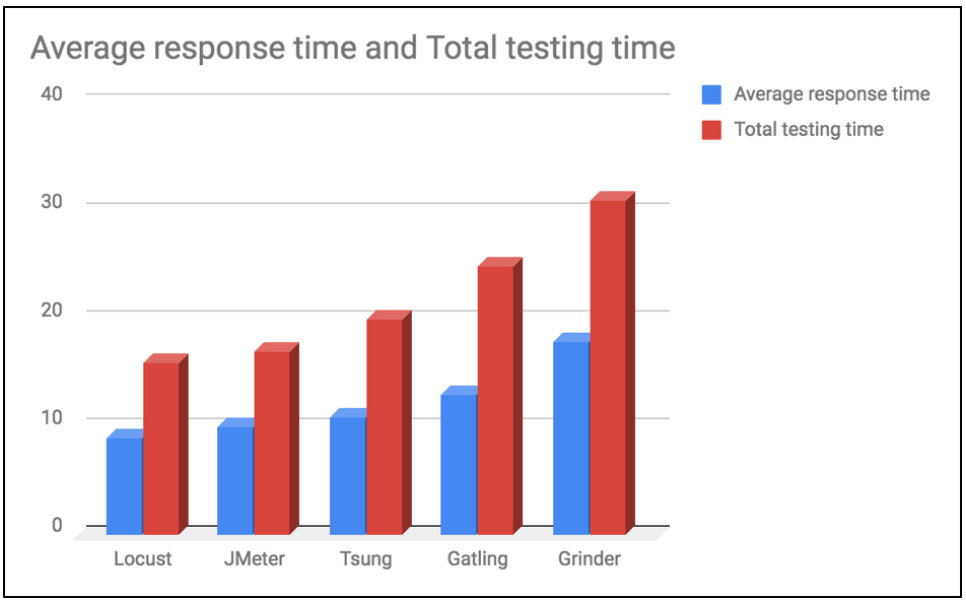


Рисунок 3.12 - Процесс нагрузочного тестирования с использованием моделей

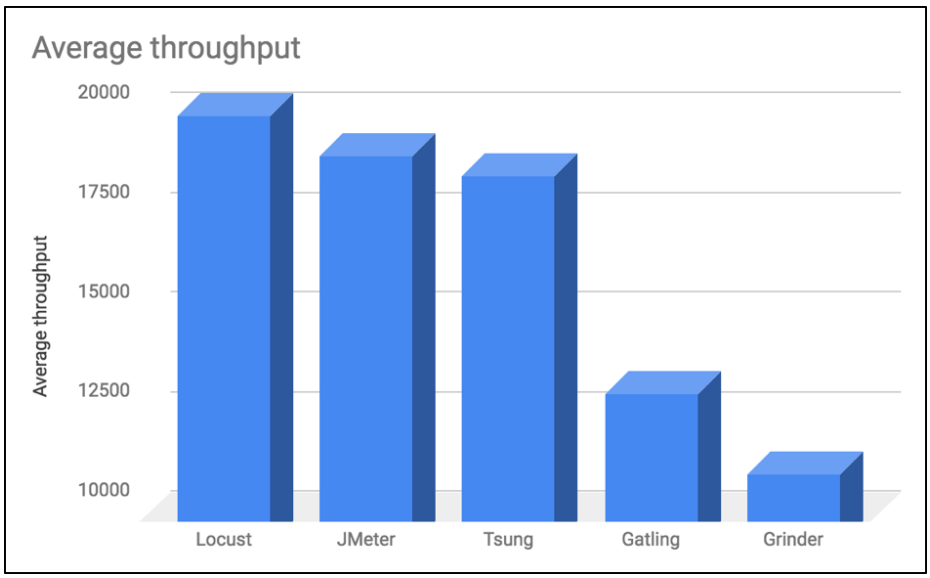


Рисунок 3.13 - Процесс нагрузочного тестирования с использованием моделей

Как показано на графиках, у Locust самое быстрое время отклика с самым высоким средним значением, за которым следуют JMeter, Tsung и Gatling. У Grinder самое медленное время с самой низкой средней пропускной способностью.

В таблице 3.2 представлено сравнение ключевых функций, предлагаемых каждым инструментом тестирования:

Таблица 3.2 – Сравнительный анализ инструментальных средств нагрузочного тестирования

Особенность	**The Grinder**	**Gatling**	**Tsung**	**JMeter**	**Locust**
Операционные системы	любой	любой	Linux / Unix	любой	любой
графический интерфейс пользователя	Только консоль	Только рекордер	нет	Полный	нет
Тест рекордер	TCP (включая HTTP)	HTTP	HTTP, Postgres	HTTP	нет
Тест Язык	Python, Clojure	Scala	XML	XML	Python
Язык Расширения	Python, Clojure	Scala	Erlang	Java, Beanshell, Javascript, Jexl	Python

Особенность	**The Grinder**	**Gatling**	**Tsung**	**JMeter**	**Locust**
Загрузить отчеты	Console	HTML	HTML	CSV, XML, встроенные таблицы, графики, плагины	HTML
протоколы	HTTP SOAP JDBC POP3 SMTP LDAP JMS	HTTP JDBC JMS	HTTP WebDAV Postgres MySQL XMPP WebSocket AMQP MQTT LDAP	HTTP FTP JDBC SOAP LDAP TCP JMS SMTP POP3 IMAP	HTTP
Мониторинг хоста	нет	нет	да	Да, с плагином PerfMon	Нет
Ограничения	Знание Python требуется для разработки и редактирования тестов. Отчеты очень простые и краткие.	Ограниченная поддержка протоколов. Требуется знание языка DSL на базе Scala. Не масштабируется.	Протестировано и поддерживается только в системах Linux.	Связанные отчеты не легко интерпретировать.	Знание Python требуется для разработки и редактирования тестов.

Применив выбранные инструментальные средства для проведения нагрузочного тестирования, были получены следующие результаты: время выборки постоянно находится между 128 и 164 мс. Географическое расстояние является наиболее важным фактором, влияющим на время выборки, если ваш сервер обладает достаточными ресурсами. В тестовом случае сервер хорошо

отреагировал на 50 пользователей, делающих запросы в течение 10 секунд. Теперь пришло время попытаться увеличить нагрузку и посмотреть, насколько хорошо система реагирует.

На следующем шаге нагрузка увеличивается до 80 за 10 секунд. Ниже приведены результаты (рис. 3.14).

Label	Sample Time(ms)	Status	Bytes	Latency
HTTP Request	953		55065	546
HTTP Request	880		55065	479
HTTP Request	941		55065	529
HTTP Request	924		55065	502
HTTP Request	815		55065	398
HTTP Request	964		55065	567
HTTP Request	885		55065	489

Рисунок 3.14 – Результаты тестирования при увеличении нагрузки

Сервер явно начинает обременяться запросами (рис. 3.15). Запустим тест JMeter для оценки использования ресурсов VPS (рис. 3.16).

```
top - 16:52:25 up 5 days, 23:18,  1 user,  load average: 0.06, 0.16, 0.13
Tasks:  74 total,   1 running,  73 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:    501868 total,   409232 used,    92636 free,    28132 buffers
```


Рисунок 3.15 – Результаты тестирования при увеличении нагрузки без использования JMeter

```
top - 16:45:57 up 5 days, 23:11,  1 user,  load average: 0.80, 0.35, 0.16
Tasks:  74 total,   3 running,  71 sleeping,   0 stopped,   0 zombie
%Cpu(s): 94.7 us,  4.7 sy,  0.0 ni,  0.3 id,  0.0 wa,  0.3 hi,  0.0 si,  0.0 st
KiB Mem:    501868 total,   410120 used,    91748 free,    28072 buffers
KiB Swap:        0 total,        0 used,        0 free.   240612 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
19233 www-data  20   0  270488  19892  12780 R 32.2  4.0   0:03.49 php5-fpm
19232 www-data  20   0  269720  19624  12772 S 31.9  3.9   0:03.83 php5-fpm
19231 www-data  20   0  271072  20664  12772 R 31.2  4.1   0:06.90 php5-fpm
 2884 mysql     20   0  821096  47956   2756 S  2.3  9.6   6:29.23 mysqld
10258 www-data  20   0   86560   2912   1020 S  1.0  0.6   0:41.01 nginx
```


Рисунок 3.16 – Результаты тестирования при увеличении нагрузки с использованием JMeter

Очевидно, что есть существенное потребление ресурсов процессора и оперативной памяти. Для удовлетворения растущих рабочих нагрузок серверы должны быть либо оптимизированы, либо нужно увеличивать ЦП. Можно также разместить базу данных на другом сервере.

Таким образом, можно поддерживать нагрузку до тех пор, пока не произойдет существенное снижение производительности. И для проверки производительности можно использовать инструментальные средства, например, JMeter имеет много применений в контексте повышения производительности и оптимизации серверов.

3.3 Тестирование производительности и надежности базы данных КИС

Рассмотрим тестирование надежности и производительности КИС. Характеристики и список оборудования для проведения нагрузочного тестирования представлен в табл. 3.3.

Таблица 3.3 - Техническое обеспечение КИС

Позиция	**Обозначение**	**Наименование и техническая характеристика**
1	Маршрутизатор	Asus RT-AC68U
2	Сервер	Xeon E5-2630, 12 x 2.60 ГГц, процессоров – 2, оперативная память – 8 ГБ, память – 2 x 500 ГБ
3	Внешнее хранилище данных	BLOB в SharePoint Foundation
4	Источник бесперебойного питания	Smart-UPS RT 3000VA RM 230V
5	Рабочие станции	Dell Precision 7510 (4K IGZO) (Precision 7000 Серия) Процессор: Intel Xeon E3-1535M v5 2.9 GHz Графический адаптер: NVIDIA Quadro M2000M, Ядро: 1038 - 1197 MHz МГц, 10.18.13.5445 - nVIDIA ForceWare 354.45, yes

Позиция	Обозначение	Наименование и техническая характеристика
		Оперативная память: 32768 Мбайт , DDR4-2133 Дисплей: 15.6 дюйм. 16:9, 3840x2160 пикс. 282 точек/дюйм, Sharp LQ156D1, IGZO IPS, глянцевое покрытие: нет Материнская плата: Intel Sunrise Point CM236 Хранение данных: Samsung SSD SM951a 512GB M.2 PCIe 3.0 x4 NVMe (MZVKV512), 512 Гбайт
6	Роутер	TP-LINK TL-WA901ND
7	Лазерные многофункционал ьные устройства	Canon i-SENSYS MF3010

Сервер приложений и БД для КИС должен быть подключен к локальной сети и иметь следующие характеристики (не ниже): Intel Xeon 3,2Mhz; 8Гб ОЗУ; 500Гб HDD; 512Мб видео.

Общими возможностями СУБД, используемой при работе КИС, являются:

- поддержка реляционной или объектно-реляционной модели базы данных;
- поддержка международного стандарта ANSI SQL-92 и выше;
- наличие средств создания индексов и кластеров данных;
- автоматическое восстановление базы данных; – поддержка сетевых протоколов TCP/IP;
- возможность контроля доступа к данным;
- централизованное управление учетными записями пользователей; - оптимизация запросов.

Тестирование осуществляется итерационно, увеличивая на каждой итерации количество одновременно работающих сессий тестов. Тестирование прекращается по одному из вышеперечисленных критериев. На каждой итерации фиксируются показатели производительности. По окончании тестирования по

снятым показателям делаются выводы о количестве одновременно работающих пользователей.

Для результативного проведения процедуры тестирования производительности КИС необходимо определенное техническое и программное обеспечение задачи. Проводить сбор и оценку характеристик функционирования СУБД не требовалось.

Определены виды применяемого тестирования производительности:

1. Определение максимальной производительности – реализуется сценарием № 2. Тест завершается, когда времена отклика превысили допустимые пределы; количество неуспешных операций увеличилось до критического (более 10%); исчерпаны системные или аппаратные ресурсы. Результатом тестирования является максимальный достигнутый уровень нагрузки (обозначается L_{max}).

2. Тест надежности – реализуется сценарием № 1. Критерием успешности тестирования является: отсутствие деградации производительности системы в ходе теста (интенсивности и времен отклика); отсутствие утечки ресурсов в течение теста.

3. Тест отказоустойчивости выполняется на уровне типичной нагрузки, который обычно устанавливается на уровне 70% от максимальной L_{max}. Критериями успешного прохождения системой теста являются:

- система сохранила доступность в функционале, не связанном с функциями смежной системы;
- система восстановилась в течение требуемого времени восстановления доступности.

Критериями успешного завершения нагрузочного тестирования являются выполнение всех запланированных тестов и получение данных мониторинга.

Для проведения нагрузочного тестирования разработаны два сценария тестирования.

1. Выполнение процедуры учета кредитования клиента в системе при одновременном участии 10 пользователей. Алгоритм процедуры представлен стандартным образом (рисунок 3.17).

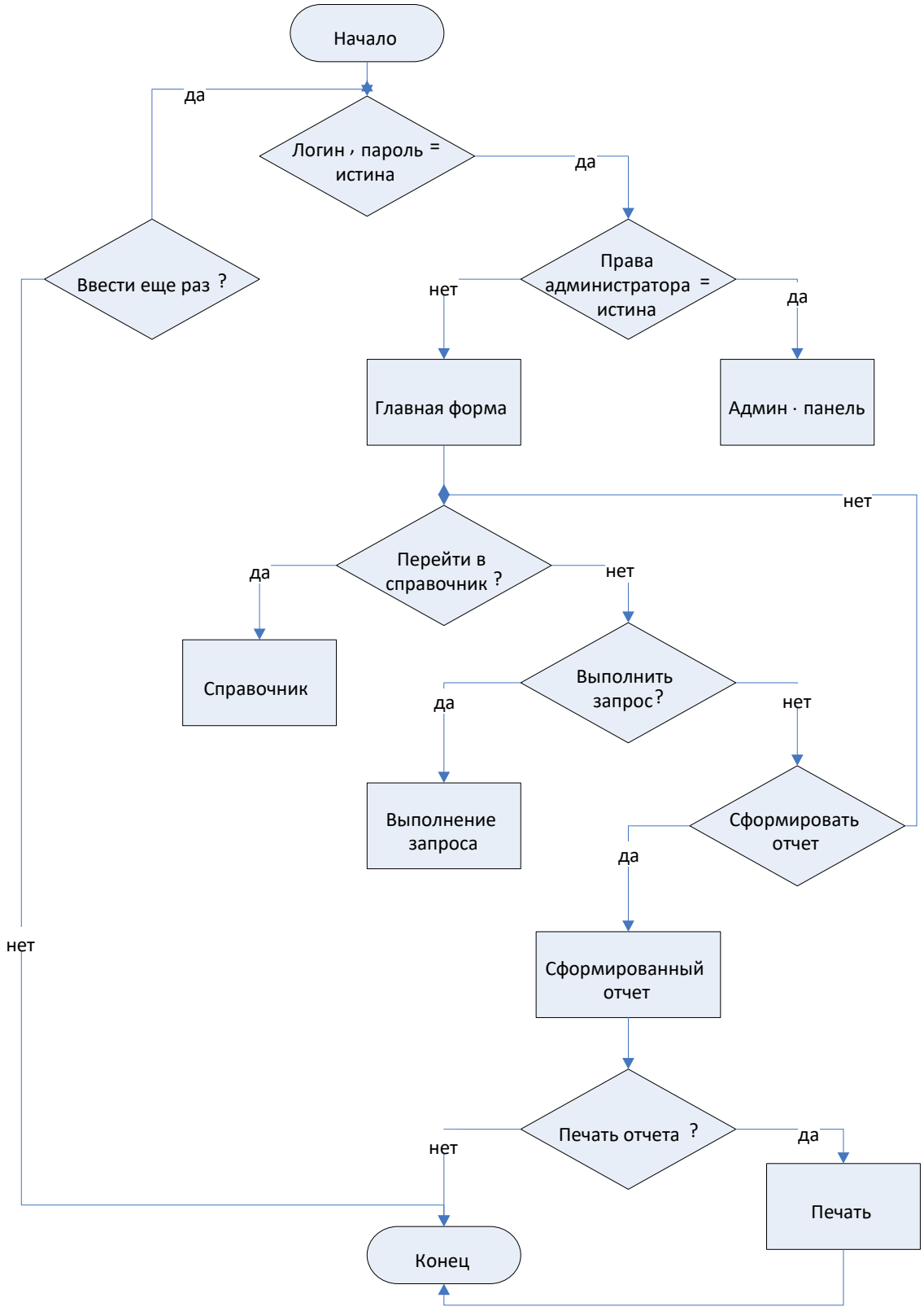


Рисунок 3.17- Алгоритм процедуры тестирования по сценарию 1

Имитация DDoS атаки (рис. 3.18). Одновременная работа в одном из справочников системы максимального числа пользователей – 30-50 ед.

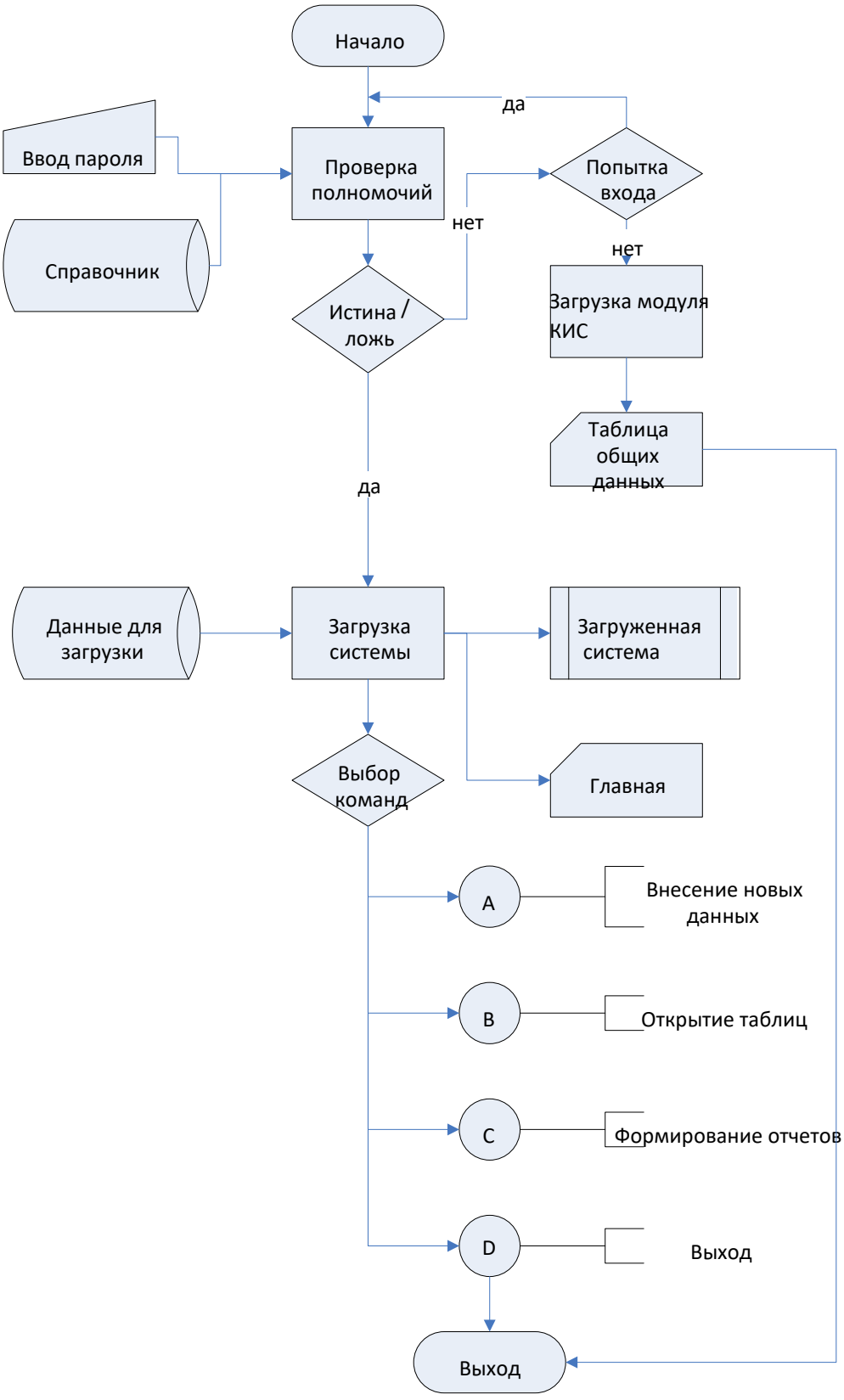


Рисунок 3.18 - Блок-схема алгоритма работы пользователя при имитации DDoS атаки

При успешности сценария количество пользователей увеличивается 10 ед. и тестирование продолжается. Определяется предел возможностей КИС.

Перед началом итерационного нагрузочного тестирования должна быть разработана программа и методика испытаний, включающая в себя:

- описание целей и задач тестирования;
- описание объекта тестирования, включая описание основных бизнес-процессов системы;
- архитектурную схему объекта тестирования;
- краткое описание подхода к тестированию и способов генерации нагрузки;
- описание профилей нагрузки, включая перечень операций, выполняемых виртуальными пользователями и внешними системами через соответствующие интерфейсы; интенсивность операций, выполняемых виртуальными пользователями и внешними системами;
- список планируемых тестов;
- типовые сценарии тестирования;
- тестовые скрипты;
- требования к тестовым данным;
- требования к производительности;
- требования к мониторингу производительности;
- метрики производительности;
- ограничения тестирования;
- ключевые показатели достижения целей; - риски проекта.

Нагрузочное тестирование КИС выполняется на заранее подготовленной базе со следующими параметрами:

1. Количество одновременно работающих сотрудников до 50.
2. Количество объектов обслуживания (юридических и физических лиц) – 10 000.
3. Количество видов обслуживания для каждого объекта – 4.
4. Количество контролируемых показателей для каждого объекта 4-5.

5. Количество показателей наработки для каждого объекта - 4-5.

После удовлетворения требований по техническому и программному обеспечению запускается нагрузочный тест, и оценивается скорость выполнения операций и прочие параметры в соответствии с предлагаемой методикой. Далее результат сравнивается с «эталонными» значениями. По результатам сравнения принимается решение о необходимости оптимизаций и выпуске обновлений.

Передача информации в КИС осуществляется путем электронной почты, по локальной сети или в форме документов. Достоверность ввода данных будет реализована при помощи счетного контроля. Формирование отчетов представляет собой механизм, основанный на использовании шаблонов отчетов. Шаблоны отчетов представляют собой параметризированные отчеты.

Аутентификация пользователей в КИС базируется на механизмах, встроенных в используемую СУБД (Microsoft SQL Server или Sybase Adaptive Server). При этом используется концепция «сквозной» аутентификации и авторизации, которая подразумевает использование единой учетной записи как на уровне системы, так и на уровне СУБД.

Для защиты информации от несанкционированного доступа и изменения лицами, обладающими административными полномочиями, в системе приняты специальные меры по разделению сфер ответственности администраторов и ограничению их полномочий.

Для определения максимально возможной нагрузки на систему необходимо получить примерную оценку реально работающих пользователей с конфигурацией. Сделать это можно по следующей методике.

Предположим, что у пользователя для работы с одним документом уходит 1 минуту. Тогда общее количество реально работающих пользователей можно получить из соотношения времени ввода документа в базу к общему времени работы с документом. При этом под общим временем работы с документом понимается суммарное время выполнения подготовительных запросов к базе по заполнению реквизитов документа, времени ввода документа в базу, а также бездействия, равному 1 минуте.

$$Rel = tsave/(tprep + tpause + tsave) \quad (3.8)$$

где

t_{prep} – суммарное время необходимой последовательности запросов к базе, имитирующей заполнение реквизитов документа;

t_{pause} – пауза в минуту;

t_{save} – время сохранения документа.

После проведения тестирования необходимо оценить отношение времени t_{save} к общему времени работы пользователя. Общее количество пользователей:

$$P = s/Rel \quad (3.9)$$

где

s – максимально достигнутое количество одновременно работающих сессий.

Рассмотрим построение марковской модели для нагрузочного тестирования КИС при работе с базами данных.

Пусть существует некоторое упорядоченное множество запросов Q к некоторой базе данных. Оно включает в себя все запросы q_i ($i = 1…n$, n – количество запросов), выполненные в БД, а также специальный «пустой запрос» q, который означает, что в данный момент времени запросов к БД не поступало, и модель переходит в начальное состояние (состояние ожидания запроса). Каждому запросу из БД сопоставляется состояние этого запроса s_i, а начальное состояние модели обозначим s_0. Далее из файла журнала анализируется заданный временной интервал T, разбитый с определенным шагом Δt на значения t_j ($j = 1…m$, m – число значений на указанном интервале). Таким образом, представленный файл журнала можно формализовать в виде множества троек вида:

$$L = \{l_j | l_j = (t_j, id, q_i)\} \quad (3.10)$$

Далее рассмотрим процесс построения марковской модели по множеству L. В начале процесса анализа модель находится в состоянии s_0. В случае, если в момент времени t_j пришел запрос q_i от клиента id, то одна тройка с указанными параметрами отмечается во множестве L, и далее не рассматривается. Модель

переходит в состояние s_i, при этом вероятность перехода в указанное состояние будет:

$$P(s_i) = \frac{n_{si}}{m} \tag{3.11}$$

где

n_{si} - количество переходов в состояние s_i;

m - общее количество разбиений временного интервала T.

В результате вычислений формируется четверка вида:

$$(s_{n-1}, s_n, q_i, p) \tag{3.12}$$

где

s_{n-1} - предыдущее состояние модели;

s_n - новое состояние модели;

q_i - запрос, вызвавший переход в состояние s_n;

p – расчетное значение вероятности.

Множество троек представляет собой марковскую модель нагрузочного тестирования КИС при работе с базами данных. Далее указанный процесс повторяется, используя в качестве базового состояния s_n. Процесс построения модели завершается в том случае, если все тройки во множестве L отмечены.

Для реализации алгоритма введем операции MARK(l_i) – отметки определенного элемента множества, операцию M(L), возвращающую как результат мощность множества L, ВЕРОЯТНОСТЬ(q_i) – результатом операции является вероятность, рассчитанная по формуле (3.11), ++ – увеличение на 1. Тогда схема алгоритма построения данной модели выглядит, как показано на рис. 3.19.

Следует отметить, что марковскую модель можно строить как для каждого клиента БД, так и для всей БД в целом.

Рассмотрим далее непосредственно процесс нагрузочного тестирования СУБД с использованием запросов КИС. Пусть существует марковская модель (рис. 3.20).

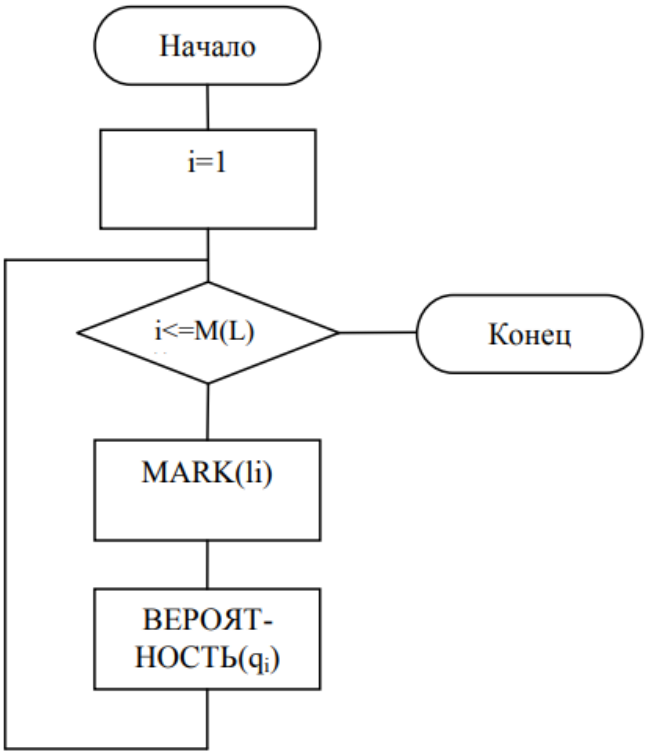


Рисунок 3.19 - Схема алгоритма построения марковской модели нагрузочного тестирования

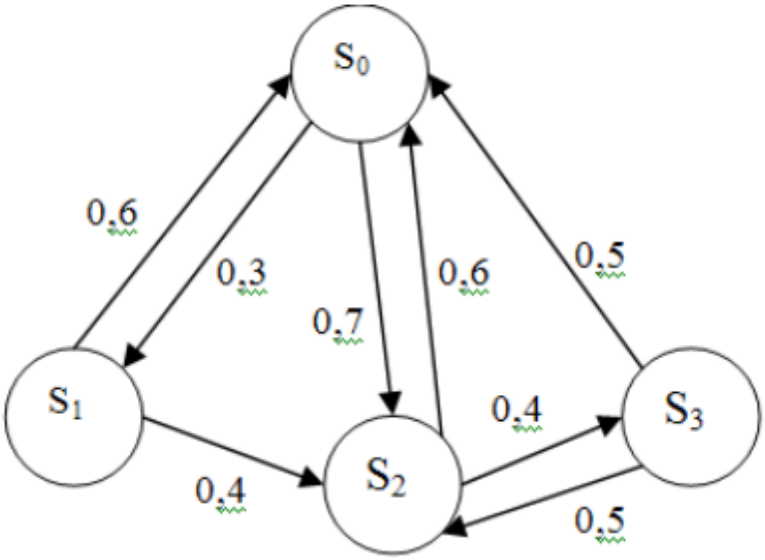


Рисунок 3.20 - Марковская модель

По итогам прохождения теста строится отношение «время ответа на конкретный запрос». Это отношение состоит из множества пар, первая компонента которых запрос, вторая – время отклика СУБД на данный запрос.

Данное отношение при постоянных параметрах настройки и структуры таблиц и запросов обладает свойством функциональности, т.е. каждому запросу соответствует свое заданное время ответа. При проведении тестов важным вопросом является получение пропорциональных результатов времени ответа на аналогичных тестах производительности. Актуальность этого вопроса связана с тем, что результаты теста зависят не только от параметров аппаратной и программной настройки сервера СУБД, но и от построения самого теста. Настройку СУБД целесообразно выполнять по результатам двух или более тестов производительности. Поэтому кроме тестирования, основанного на марковских моделях поведения клиентов СУБД, целесообразно проводить тестирование одним или несколькими синтетическими тестами, структура запросов которых в наибольшей степени совпадают со структурой запросов КИС. Для выяснения степени этого соответствия, построенные ранее марковские модели, необходимо дополнить информацией о структуре запросов КИС.

Тестирование производительности позволяет определить степень готовности системы к внештатным ситуациям (отказ оборудования, DDoS-атаки), уровень ее надежности и способность к самовосстановлению. Также нагрузочные испытания помогают разработать комплекс адекватных мер для повышения производительности системы, ее устойчивости и защищенности корпоративного окружения.

3.4 Сценарий нагрузочного тестирования и формирование критериев на доработку

В качестве примера произведем нагрузочное тестирование КИС под 30 пользователями одновременно, которые последовательно входят в систему с интервалом 3 секунды. Динамика подачи нагрузки представлена на рисунке 3.21.

Рисунок 3.21 – Динамика подачи нагрузки

Все 30 пользователей начинают работать с системой спустя 90 секунд с момента начала нагрузочного теста. Распределение времени отклика по транзакциям относительно начала нагрузочного теста представлено на рисунках 3.22 и 3.23.

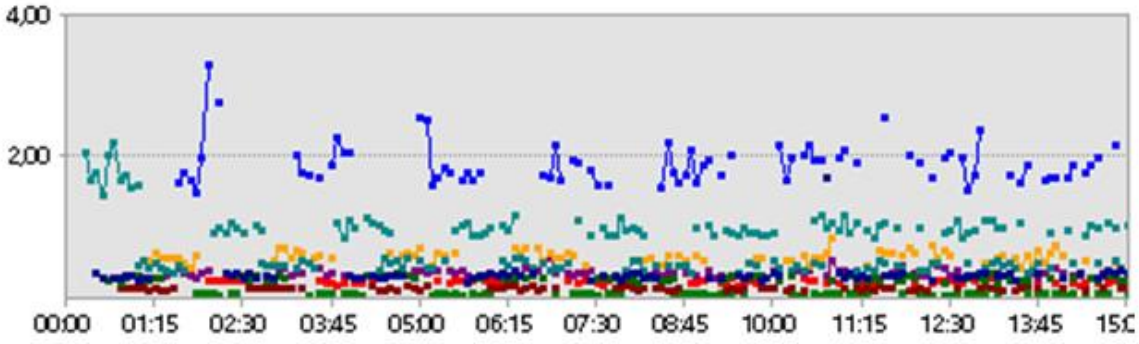


Рисунок 3.22 – Время отклика Системы

Экземпляр	Цвет	Минимум	Максимум	Средн.
01 Авторизоваться в системе		0,86	2,23	1,09
02 Создать новый выпуск		0,24	1,75	0,34
03 Открыть вкладку Параметры выпусков		0,23	0,38	0,31
04 Выбрать выпуск Москва 44		0,12	0,20	0,16
05 Перейти в раздел Оценка эффективности		0,36	0,60	0,48
06 Создать новый сценарий выкупа		0,46	0,87	0,61
07 Ввести данные, сохранить и рассчитать		0,29	0,59	0,39
08 Открыть раздел Мониторинг и анализ		1,52	3,34	1,95
09 Открыть калькулятор и рассчитать цену		0,054	0,092	0,075
10 Выйти из системы		0,19	0,33	0,26

Рисунок 3.23 – Время отклика Системы (легенда)

В процессе выполнения сценария все транзакции были выполнены успешно. В таблице 3.4 приведены данные по распределению времени отклика.

Таблица 3.4. Статистика выполнения сценария

Транзакция	**Минимум (сек)**	**Среднее (сек)**	**Максимум (сек)**
01. Ввести неверный логин и пароль и проверить аут	0,012	0,013	0,017
02. Войти в систему под паролем	0,70	4,01	63,0
03. Открыть раздел «Ввод данных»	0,10	2,89	32,1
04. Открыть Паспорт	0,12	0,17	1,43
05. Открыть Паспорт.	0,11	0,15	0,71
06. Перейти в таблицу 1	0,054	0,37	25,4
07. Ввести 50 значений в таблицу и Сохранить	0,080	1,19	32,4
08. Перейти на уровень ОМ	0,087	0,54	8,94
09. Перейти в таблицу 4	0,12	0,36	19,6
10. Ввести 50 значений в таблицу и Сохранить	0,12	0,43	27,9
11. Перейти в раздел Печатные формы	0,15	0,43	23,9
12. Выбрать программу и версию	0,16	0,21	0,28
13. Выбрать таблицу 1 и Сформировать	5,84	6,33	7,29
14. Выбрать таблицу 4 и Сформировать	1,05	1,28	1,72
15. Выбрать таблицу 2 и Сформировать	9,67	14,2	43,9
16. Выйти из системы	0,028	0,057	0,13
17. Войти в систему под admin	1,07	3,6	33,7
18. Открыть вкладку «Мониторинг»	0,15	14,1	32,4
19. Открыть раздел «Свод. отчет в Прав.-о выполнен	0,098	7,13	30,7
20. Выбрать 3 квартал 2018 года	2,05	3,81	20,7
21. Сформировать Пояснительную записку	0,12	0,14	0,22
22. Открыть раздел «Сводный отчет»	2,07	2,28	2,61
23. Выйти из системы	0,033	0,043	0,070

По результатам таблицы 3.4 видно, что время отклика системы удовлетворяет требованиям производительности при нагрузке 30 пользователей. Наименее производительными являются транзакции: «13. Выбрать таблицу 1 и Сформировать», «15. Выбрать таблицу 2 и Сформировать», «Открыть вкладку «Мониторинг», рекомендуется обратить внимание на их оптимизацию.

В процессе нагрузочного тестирования необходимо снимать счетчики производительности с сервера приложений. Результаты представлены на рисунках 3.24-3.25.

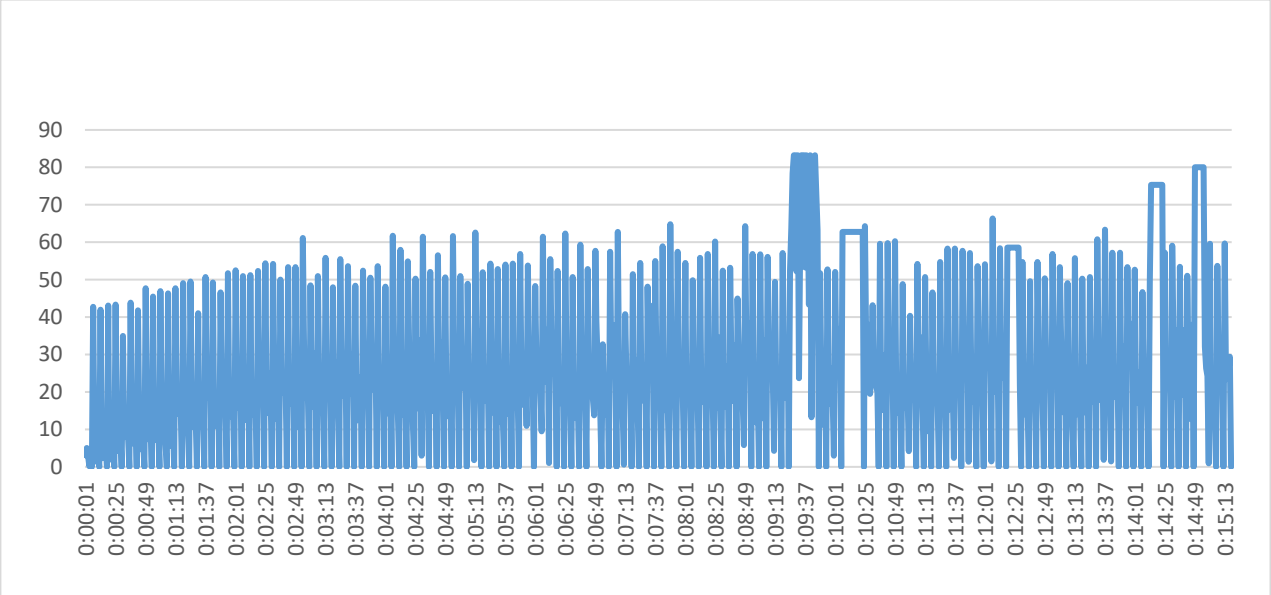


Рисунок 3.24 – Потребление ресурсов CPU сервера приложений

Рисунок 3.25 – Потребление памяти сервера приложений

Также необходимо снимать счетчики производительности с сервера базы данных (БД). Результаты представлены на рисунках 3.26-3.27.

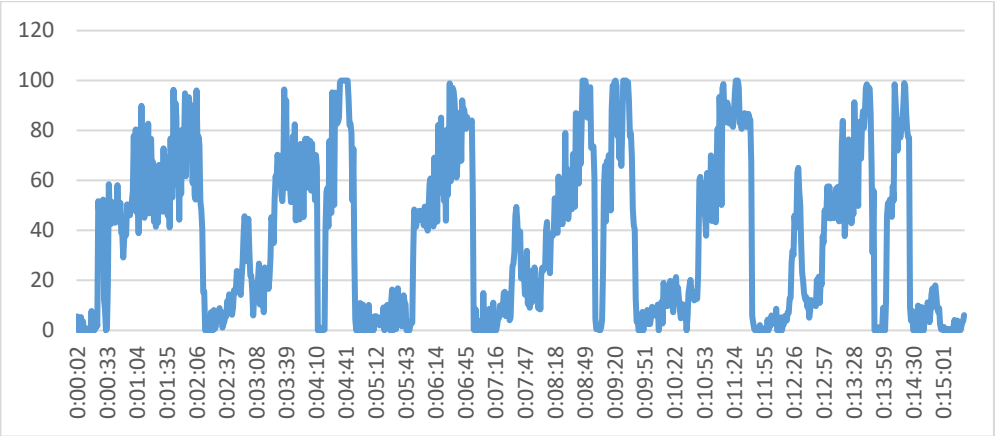


Рисунок 3.26 – Потребление ресурсов CPU сервера БД

Рисунок 3.27 – Потребление памяти сервера БД

Из графиков видно, что на протяжении всего выполнения сценария наблюдается средняя загрузка CPU сервера приложении 60% с пиками до 85%, среднее сервера БД 60% с пиками до 100%. Потребление памяти сервером БД и Приложений находится на среднем уровне 10,4-11,3 гб и 56,2-56,8 гб.

В результате проведенного нагрузочного тестирования можно сделать следующие выводы:

– CPU сервера БД, вероятно, не является узким местом, ограничивающим производительность системы;

– CPU сервера приложений, вероятно, является узким местом, ограничивающим производительность системы;

– потребление памяти сервера приложений и сервера БД не является узким местом, ограничивающим производительность системы;

– на основе анализа счетчиков производительности работа дисковой подсистемы не является узким местом; аналогично для загрузки клиентской станции.

– в качестве рекомендации разработчикам системы следует обратить внимание на оптимизацию наименее производительных транзакций «13. Выбрать таблицу 1 и Сформировать», «15. Выбрать таблицу 2 и Сформировать», «Открыть вкладку «Мониторинг».

3.5 Интеграционное тестирование корпоративной информационной системы

Проведено интеграционное тестирование с использованием платформы Citrus Framework.

Citrus - это интегрированная среда тестирования, написанная на Java, которая тестирует КИС на предмет соответствия среде клиента. Инструмент имитирует окружающие системы через различные порты (http, JMS, TCP / IP, SOAP, …), чтобы выполнить автоматическое сквозное тестирование варианта использования.

Составляя возможные сценарии, осуществляется проверка КИС, чтобы иметь возможность отправлять сообщения по разным протоколам и выполнять всю корреляцию сообщений, которые поступают и в конечном итоге прибывают в целевой пункт назначения.

Чтобы смоделировать один из этих сценариев, был использован Citrus Framework. Для проверки его работы, была смоделирована структура каталогов (рис. 3.28):

– src / citrus / java - местоположение сгенерированных тестовых примеров Testng;

– src / citrus / resoruces - расположение всей конфигурации, необходимой для фреймворка;

– src / citrus / tests- расположение всех определенных тестовых случаев;

– lib - расположение всех библиотек зависимостей;

– log - расположение, в котором Citrus Framework будет хранить логи (фрагменты полезной нагрузки);

– build.xml- скрипт сборки ant.

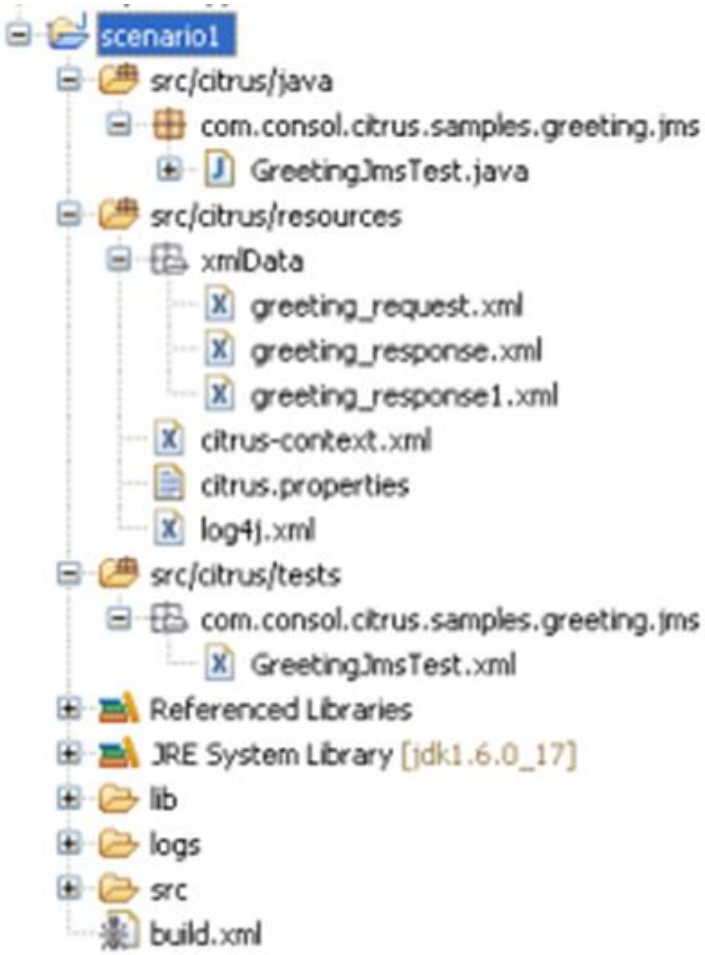


Рисунок 3.28 – Структура каталогов

На рис. 3.29 представлен пример используемого build.xml.

```xml
<project name="greetings" basedir="." default="citrus.run.tests">

    <property file="src/citrus/resources/citrus.properties"/>

    <path id="citrus-classpath">
        <pathelement path="src/citrus/java"/>
        <fileset dir="lib">
            <include name="*.jar"/>
        </fileset>
    </path>

    <typedef resource="citrustasks" classpath="lib/citrus-ant-tasks-1.1-SNAPSHOT.jar"/>

    <target name="compile.tests">
        <javac srcdir="src/citrus/java" classpathref="citrus-classpath"/>
        <javac srcdir="src/citrus/tests" classpathref="citrus-classpath"/>
    </target>

    <target name="citrus.run.tests" depends="compile.tests" description="Runs all Citrus
        <citrus suitename="${testsuite.name}" package="com.consol.citrus.samples.*"/>
    </target>

    <target name="citrus.run.single.test" depends="compile.tests" description="Runs a si
        <touch file="test.history"/>

        <loadproperties srcfile="test.history"/>

        <echo message="Last test executed: ${last.test.executed}"/>

        <input message="Enter test name:" addproperty="testclass" defaultvalue="${last.t

        <propertyfile file="test.history">
            <entry key="last.test.executed" type="string" value="${testclass}"/>
        </propertyfile>

        <citrus suitename="citrus-samples" test="${testclass}"/>
    </target>

    <target name="create.test" description="Creates a new empty test case">
        <input message="Enter test name:" addproperty="test.name"/>
        <input message="Enter test description:" addproperty="test.description" defaultv
        <input message="Enter author's name:" addproperty="test.author" defaultvalue="${
        <input message="Enter package:" addproperty="test.package" defaultvalue="${defau
        <input message="Enter framework:" addproperty="test.framework" defaultvalue="tes

        <java classname="com.consol.citrus.util.TestCaseCreator">
            <classpath refid="citrus-classpath"/>
            <arg line="-name ${test.name} -author ${test.author} -description ${test.des
        </java>
    </target>

    <target name="create.html.doc" description="Creates test documentation in html">
        <mkdir dir="target"/>

        <java classname="com.consol.citrus.doc.HtmlTestDocGenerator">
            <classpath refid="citrus-classpath" />

            <arg value="src/citrus/tests"/>
            <arg value="target/CitrusTests.html"/>
            <arg value="Citrus Test Documentation"/>
            <arg value="logo.png"/>
            <arg value="Overview"/>
        </java>

        <copy todir="target" file="src/citrus/resources/logo.png"/>

        <zip destfile="target/CitrusTests.zip">
            <fileset dir="target">
                <include name="CitrusTests.html"/>
                <include name="logo.png"/>
            </fileset>
        </zip>
    </target>

    <target name="create.xls.doc" description="Creates test documentation in excel">
        <mkdir dir="target"/>

        <java classname="com.consol.citrus.doc.ExcelTestDocGenerator">
            <classpath refid="citrus-classpath" />

            <arg value="src/citrus/tests"/>
            <arg value="CitrusTests"/>
            <arg value="Citrus Test Documentation"/>
            <arg value="Citrus Testframework"/>
            <arg value="ConSol* Software GmbH"/>
        </java>
    </target>
</project>
```

Рисунок 3.29 – Пример используемого build.xml

ПО КИС по сценарию получает новое сообщение в очереди. Какой-то процесс получает сообщение, выполняет его преобразование / проверку и передает его. В конце концов, новое сообщение будет отправлено в другую очередь.

Конфигурация по умолчанию описывает использование:

<citrus:jms-message-sender id="getOrdersRequestSender" destination-name="testJMSServer/citrus_queue_in"/>

Нужно сконфигурировать src / citrus / tests / com / consol / citrus / samples /reeting / jms / GreetingJmsTest.xml (рис. 3.30).

```
<?xml version="1.0" encoding="UTF-8"?>
<spring:beans xmlns="http://www.citrusframework.org/schema/testcase" xmlns:spring="http:,
    <testcase name="GreetingJmsTest">
        <meta-info>
            <author>Eric Elzinga</author>
            <creationdate>2010-03-25</creationdate>
            <status>FINAL</status>
            <last-updated-by>Eric Elzinga</last-updated-by>
            <last-updated-on>2010-03-28T00:00:00</last-updated-on>
        </meta-info>

        <variables>
            <variable name="correlationId" value="citrus:randomNumber(10)"></variable>
            <variable name="user" value="Eric"></variable>
        </variables>

        <actions>
            <send with="sendGreeting">
            <message>
                <resource file="classpath:xmlData/greeting_request.xml" />
            </message>
            <header>
                <element name="Operation" value="sayHello"/>
                <element name="CorrelationId" value="${correlationId}"/>
            </header>
        </send>

        <receive with="receiveGreeting">
        <selector>
            <value>CorrelationId = '${correlationId}'</value>
        </selector>
        <message>
          <resource file="classpath:xmlData/greeting_response.xml" />
          <ignore path="//tns:GreetingRequestMessage/tns:Text" />
        </message>
        <header>
            <element name="Operation" value="sayHello"/>
            <element name="CorrelationId" value="${correlationId}"/>
        </header>
    </receive>
</actions>
</testcase>
</spring:beans>
```

Рисунок 3.30 – Файл конфигурации

Чтобы включить полезную нагрузку теста, можно использовать следующие команды:

```
<message>
    <resource file="classpath:xmlData/greeting_request.xml" />
</message>
```

или же

```
<message>
    <data>
        <![CDATA[
           <tns:GreetingRequestMessage xmlns:tns="http://www.citrusframework.org/samp
               <tns:CorrelationId>${correlationId}</tns:CorrelationId>
               <tns:Operation>sayHello</tns:Operation>
               <tns:User>${user}</tns:User>
               <tns:Text>Hello Citrus!</tns:Text>
           </tns:GreetingRequestMessage>
        ]]>
    </data>
</message>
```

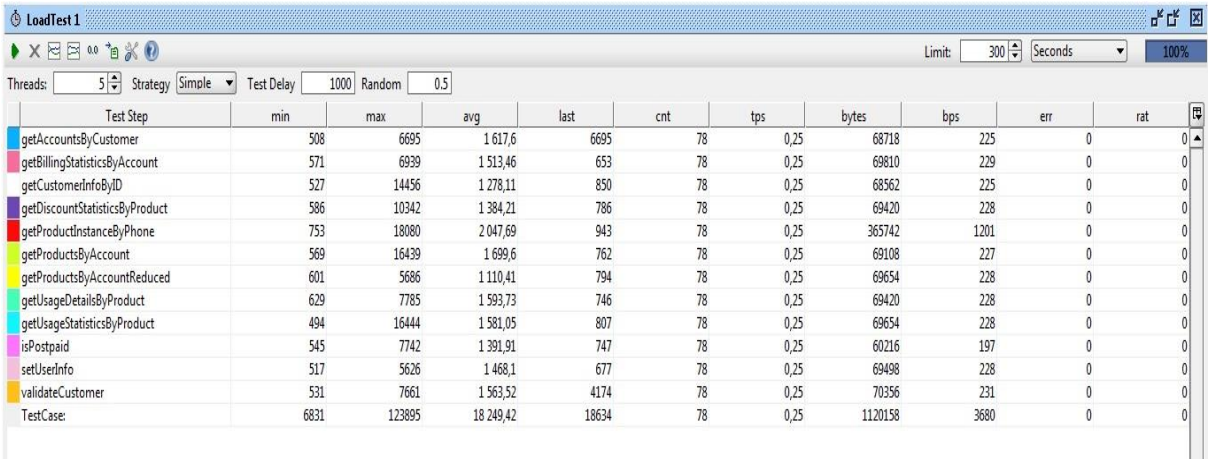
Таким образом, была проведена проверка сообщений XML и получаем результат, представленный на рисунке 3.31.

```
[citrus] 4547   INFO  port.LoggingReporter| TEST FINISHED: GreetingJmsTest
[citrus] 4547   INFO  port.LoggingReporter| ------------------------------------------
[citrus] 4563   INFO  port.LoggingReporter| FINISH TESTSUITE citrus-samples-greeting
[citrus] 4563   INFO  port.LoggingReporter| ------------------------------------------
[citrus] 4563   INFO  port.LoggingReporter| __________________________________________
[citrus] 4563   INFO  port.LoggingReporter|
[citrus] 4563   INFO  port.LoggingReporter| CITRUS TEST RESULTS
[citrus] 4563   INFO  port.LoggingReporter|
[citrus] 4563   INFO  port.LoggingReporter|  GreetingJmsTest .........................
[citrus] 4563   INFO  port.LoggingReporter|
[citrus] 4563   INFO  port.LoggingReporter| Total number of tests: 1
[citrus] 4563   INFO  port.LoggingReporter| Skipped: 0 (0.0%)
```

Рисунок 3.31 – Результат работы теста

Данным примером была продемонстрирована возможность проверки – можно ли поместить сообщение в очередь и получить ответ обратно в очередь; проверяет ли ответ xml соответствие предоставленному определению схемы (xsd).

На рисунке 3.32 представлены результаты тестирования интеграции с помощью soapUI.



Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
getAccountsByCustomer	508	6695	1 617,6	6695	78	0,25	68718	225	0	0
getBillingStatisticsByAccount	571	6939	1 513,46	653	78	0,25	69810	229	0	0
getCustomerInfoByID	527	14456	1 278,11	850	78	0,25	68562	225	0	0
getDiscountStatisticsByProduct	586	10342	1 384,21	786	78	0,25	69420	228	0	0
getProductInstanceByPhone	753	18080	2 047,69	943	78	0,25	365742	1201	0	0
getProductsByAccount	569	16439	1 699,6	762	78	0,25	69108	227	0	0
getProductsByAccountReduced	601	5686	1 110,41	794	78	0,25	69654	228	0	0
getUsageDetailsByProduct	629	7785	1 593,73	746	78	0,25	69420	228	0	0
getUsageStatisticsByProduct	494	16444	1 581,05	807	78	0,25	69654	228	0	0
isPostpaid	545	7742	1 391,91	747	78	0,25	60216	197	0	0
setUserInfo	517	5626	1 468,1	677	78	0,25	69498	228	0	0
validateCustomer	531	7661	1 563,52	4174	78	0,25	70356	231	0	0
TestCase:	6831	123895	18 249,42	18634	78	0,25	1120158	3680	0	0

Рисунок 3.32- Результаты нагрузочного тестирования Wizard API

Для тестирования было определено 1000 запросов, с одновременной отправкой 5 запросов на сервер. Среднее время полной загрузки страницы при работе одновременно 50 человек составляет около 25 секунд, что является в пределах допустимости. Скорость передачи данных между системами определяется в интервалах: до 30 сек – высокая, 30 - 50 – средняя, от 50 – низкая.

Таким образом, на данном примере была продемонстрирована возможность приемочного тестирования показать, что все модули интегрированы между собой и имеют хорошее взаимодействие, при работе не обнаружены дефекты. И это подтверждается опытным путем.

3.6 Аналитические показатели тестирования безопасности корпоративных информационных систем

КИС представляет собой сложную структуру, объединяющую в себе различные сервисы, необходимые для функционирования компании. Эта структура постоянно меняется: появляются новые элементы, изменяется конфигурация существующих. По мере изменения и увеличения КИС все более важной и актуальной задачей становятся обеспечение информационной безопасности и защита критически важных для компании ресурсов.

С целью выявления недостатков защиты различных компонентов и определения потенциальных атак на информационные ресурсы, проводится анализ защищенности. Наиболее эффективный способ анализа - тестирование на проникновение, в ходе которого моделируется реальная атака злоумышленников. Такой подход позволяет объективно оценить уровень защищенности корпоративной инфраструктуры и понять, могут ли противостоять атакам применяемые в компании средства защиты.

Для выполнения анализа защищенности КИС приведем обзор наиболее распространенных недостатков безопасности, практические примеры их эксплуатации и описание вероятных векторов атак, а также рекомендации по повышению уровня защищенности.

Для подведения статистики была сформирована итоговая выборка ряда российских и зарубежных компаний из различных отраслей экономики (из числа тех компаний, которые разрешили использовать обезличенные данные), при этом большую часть составили промышленные, финансовые и транспортные компании (рис. 3.33).

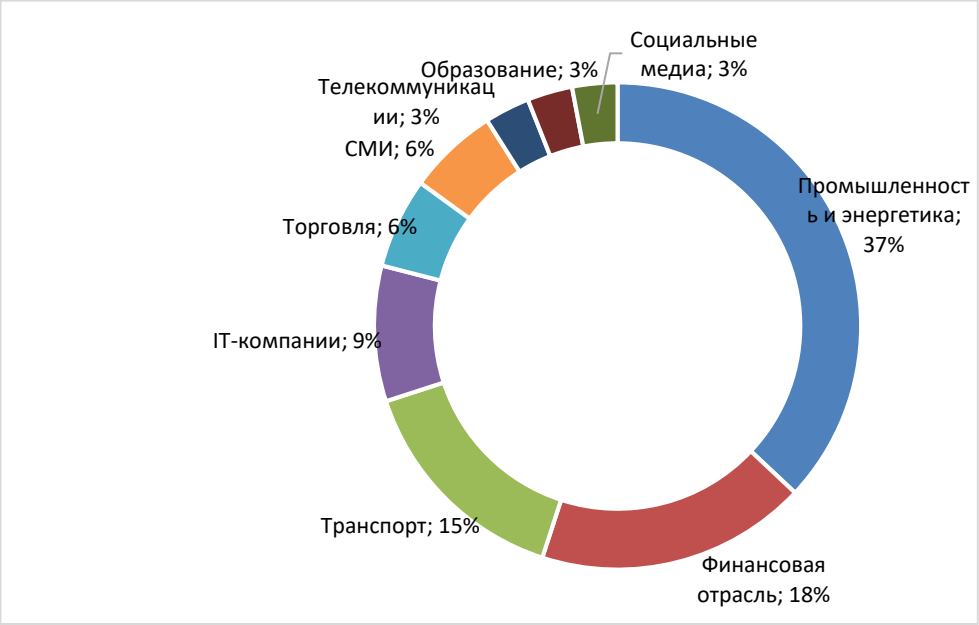


Рисунок 3.33 - Распределение исследованных систем по отраслям экономики

Анализ защищенности КИС проводится путем внешнего и внутреннего тестирования на проникновение. Чтобы сформировать корректную оценку уровня защищенности, воссоздаются условия, максимально приближенные к условиям реальной атаки. В ходе внешнего тестирования моделируются действия потенциального злоумышленника, который не обладает привилегиями в рассматриваемой системе и действует из интернета. В этом случае перед экспертами ставится задача преодолеть сетевой периметр и получить доступ к ресурсам локальной сети. Внутреннее тестирование подразумевает, что нарушитель действует из сегмента локальной сети, а его целью является контроль над инфраструктурой или над отдельными критически важными ресурсами, которые определяет заказчик. Комплексное тестирование на проникновение подразумевает оба вида работ.

Для некоторых компаний выполнялись анализ защищенности беспроводных сетей и оценка осведомленности персонала в вопросах информационной безопасности.

Рисунок 3.34 - Виды проведенных работ

При анализе защищенности эксперты выявляют различные уязвимости и недостатки механизмов защиты, которые можно разделить на четыре категории:

- недостатки конфигурации;
- отсутствие обновлений безопасности;
- уязвимости в коде веб-приложений;
- недостатки парольной политики.

Каждой уязвимости присваивается уровень риска, который рассчитывается в соответствии с системой классификации CVSS 3.0. Практически во всех системах были обнаружены критически опасные уязвимости, в основном связанные с недостатками парольной политики.

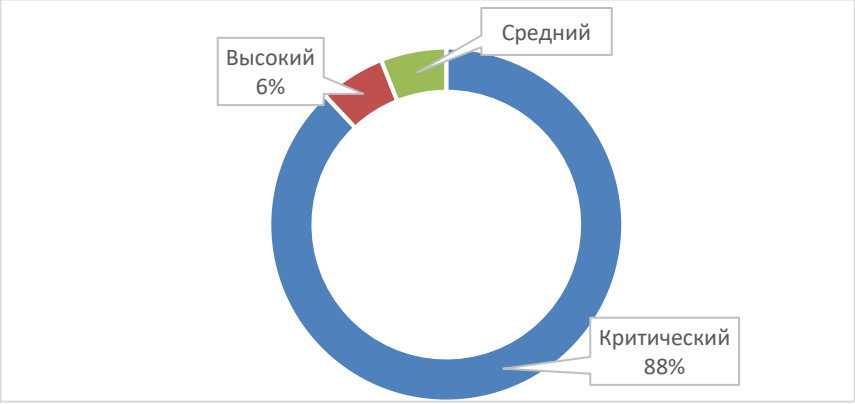


Рисунок 3.35 - Максимальный уровень опасности уязвимостей (доля систем)

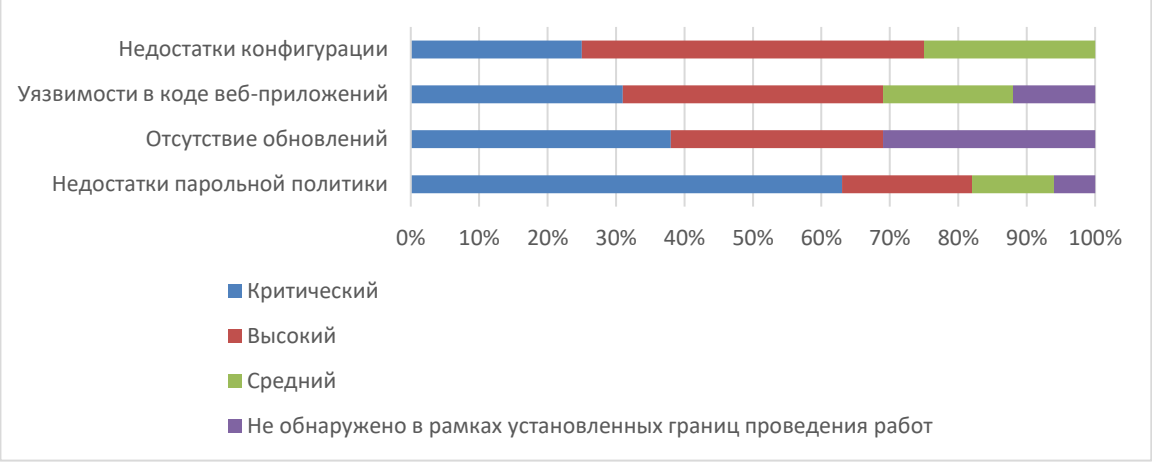


Рисунок 3.36 - Максимальный уровень опасности уязвимостей (доля систем)

Важно учитывать, что работы по тестированию на проникновение проводятся методом черного ящика, поэтому невозможно выявить все уязвимости, существующие в системе. В инфраструктуре каждой компании могли присутствовать недостатки защиты, обусловленные отсутствием своевременного обновления ПО, уязвимостями в коде веб-приложений и использованием словарных паролей, которые не были обнаружены в ходе анализа. Целью тестирования на проникновение является не поиск всех без исключения недостатков системы, а получение объективной оценки уровня ее защищенности от атак нарушителей.

В большинстве случаев проникнуть во внутреннюю сеть можно несколькими способами. В среднем на одну систему приходится два вектора, а максимальное число векторов проникновения, обнаруженных в одной системе, - пять. По статистике в половине исследуемых компаний существует способ преодолеть сетевой периметр всего за один шаг; как правило, он заключался в эксплуатации уязвимости в веб-приложении.

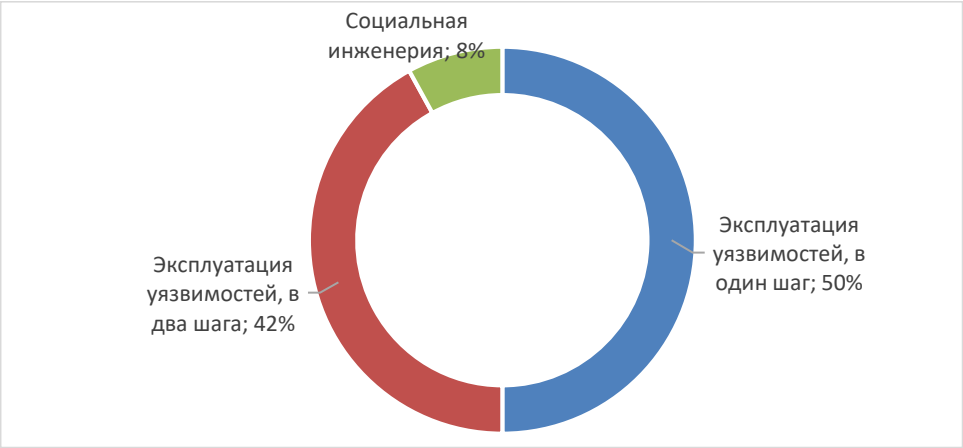


Рисунок 3.36 - Кратчайший путь преодоления сетевого периметра (доля систем)

Три четверти векторов оказались связаны с недостаточной защитой веб-приложений: это основная проблема на сетевом периметре. При этом, если вектор состоит из нескольких шагов, на каждом шаге могут

эксплуатироваться уязвимости разного типа. Типовой сценарий атаки представляет собой подбор словарной учетной записи пользователя КИС и последующая эксплуатация уязвимости, возникшей из-за ошибок в коде веб-приложения, например возможности загрузки на сервер произвольных файлов.

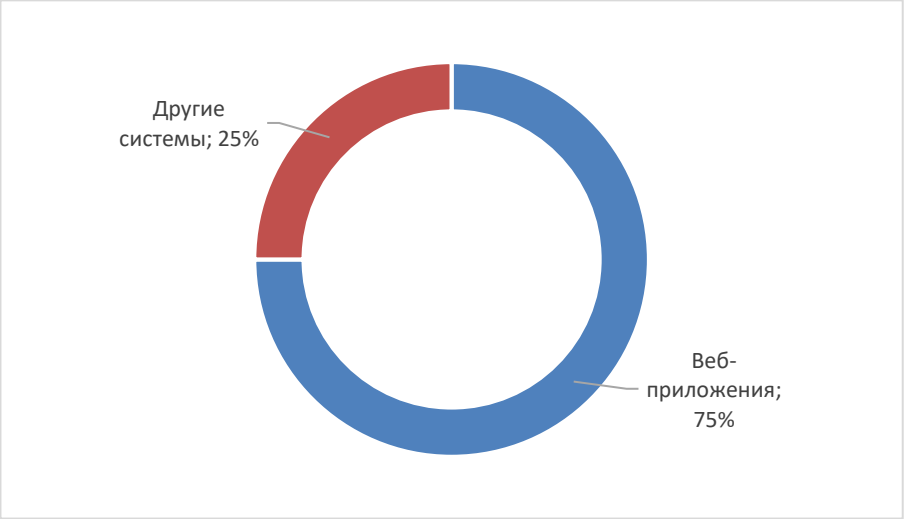


Рисунок 3.37 - Векторы проникновения во внутреннюю сеть

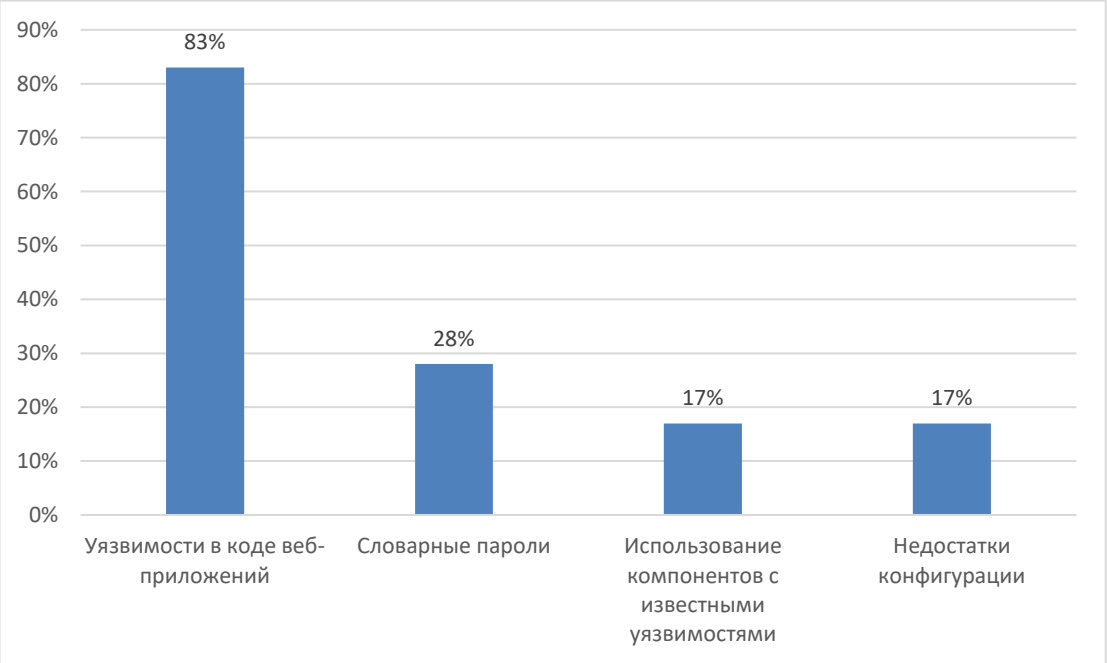


Рисунок 3.38 - Уязвимости веб-приложений, позволившие преодолеть сетевой периметр (доли векторов)

Рекомендуется регулярно проводить анализ защищенности веб-приложений. Чем сложнее веб-приложение и чем больше у него различных функций, тем выше вероятность того, что разработчики допустили в коде ошибку, которая позволит злоумышленнику провести атаку. Частично такие ошибки выявляются в рамках тестирования на проникновение, но наибольшее их число может быть выявлено только при проверке приложения методом белого ящика, подразумевающим анализ исходного кода. Для исправления уязвимостей обычно требуется внести изменения в код, на что может потребоваться значительное время. Чтобы сохранить непрерывность бизнес-процессов, рекомендуется применять межсетевой экран уровня приложений (web application firewall), который не позволит эксплуатировать уязвимость, пока ее не устранили, а также защитит от новых и еще не найденных уязвимостей.

Прочие векторы состояли преимущественно в подборе словарных паролей к различным системам — Outlook Web App (OWA), VPN-серверам и рабочим станциям, а также в использовании недостатков конфигурации сетевого оборудования. Преодоление периметра потенциально возможно и через устаревшие версии ПО, которые содержат уязвимости, позволяющие получить контроль над сервером. Для многих таких уязвимостей есть общедоступные эксплойты, но их эксплуатация может нарушить работу систем, поэтому заказчики, как правило, не соглашаются на проведение подобных проверок в рамках тестирования на проникновение.

В качестве примера эксплуатации возьмем словарные пароли пользователей. В ходе тестирования на проникновение обнаружено, что для доступа к сервису OWA используется доменная учетная запись test:test1234. Подключившись к OWA, была загружена автономная адресная книга (Offline Address Book), где содержатся идентификаторы пользователей домена. Подобрав словарный пароль к учетной записи одного из

пользователей, можно подключиться к шлюзу удаленных рабочих столов (RDG) и по протоколу RDP получить доступ к компьютеру сотрудника компании и внутренней сети.

Для полного представления тестирования уязвимости КИС проанализируем интерфейс управления оборудованием, доступный из внешних сетей, словарные пароли пользователей.

Один из распространенных вариантов проведения успешных атак в рамках тестирования — обнаружение на сетевом периметре интерфейсов систем, которые должны быть доступны исключительно из внутренней сети. Важно правильно определять границы сетевого периметра и следить за состоянием защищенности каждого компонента системы.

По итогам статистики десятка наиболее распространенных уязвимостей на сетевом периметре мало изменяется из года в год. В ряде компаний широко распространено использование открытых протоколов передачи данных, в том числе для доступа к интерфейсам администрирования. Злоумышленник может перехватить учетные данные, передаваемые по открытым протоколам без использования шифрования, и получить доступ к соответствующим ресурсам. Более чем в половине исследуемых Систем внешнему нарушителю доступны интерфейсы удаленного доступа, управления оборудованием и подключения к СУБД.

В качестве рекомендаций можно предложить следующие:

- ограничить количество сервисов на сетевом периметре, убедиться в том, что открытые для подключения интерфейсы действительно должны быть доступны всем интернет-пользователям;
- регулярно проводить инвентаризацию ресурсов, доступных для подключения из интернета. Уязвимости могут появиться в любой момент, поскольку конфигурация инфраструктуры постоянно меняется, в ней

появляются новые узлы, новые системы, и не исключены ошибки администрирования;

- отказаться от использования простых и словарных паролей, разработать строгие правила для корпоративной парольной политики и контролировать их выполнение.

Рисунок 3.39 - Наиболее распространенные уязвимости на сетевом периметре (доля систем)

На ресурсах сетевого периметра часто хранятся в открытом виде важные данные, которые помогают злоумышленнику развить атаку. Это могут быть резервные копии веб-приложений, конфигурационная информация о системе, учетные данные для доступа к критически важным ресурсам или идентификаторы пользователей, к которым злоумышленник может подобрать пароль.

Во избежание выявленных угроз рекомендуется убедиться, что в открытом виде (например, на страницах веб-приложения) не хранится

чувствительная информация, представляющая интерес для злоумышленника. К такой информации могут относиться учетные данные для доступа к различным ресурсам, адресная книга компании, содержащая электронные адреса и доменные идентификаторы сотрудников, и т. п. Если у компании не хватает ресурсов, чтобы выполнить такие проверки собственными силами, то рекомендуется привлекать сторонних экспертов для тестирования на проникновение. Актуальной остается и проблема несвоевременного обновления ПО.

Далее представлены результаты внутренних тестов на проникновение. Для этого во всех исследуемых системах необходим полный контроль над внутренней инфраструктурой. В среднем для этого требуется четыре шага. Типовой вектор атаки строится на подборе словарных паролей и восстановлении учетных записей из памяти ОС с помощью специальных утилит. Повторяя эти шаги, злоумышленник перемещается внутри сети от одного узла к другому вплоть до обнаружения учетной записи администратора домена.

Рекомендуется обеспечить защиту инфраструктуры от атак, направленных на восстановление учетных записей из памяти операционной системы (ОС). Для этого на всех рабочих станциях привилегированных пользователей, а также на всех узлах, к которым осуществляется подключение с использованием привилегированных учетных записей, установить Windows версии выше 8.1 (на серверах — Windows Server 2012 R2 или выше) и включить привилегированных пользователей домена в группу Protected Users. Кроме того, можно использовать современные версии Windows 10 на рабочих станциях и Windows Server 2016 на серверах, в которых реализована система Remote Credential Guard, позволяющая изолировать и защитить системный процесс lsass.exe от несанкционированного доступа. Обеспечить дополнительную защиту

привилегированных учетных записей (в частности администраторов домена). Хорошей практикой является использование двухфакторной аутентификации.

Рисунок 3.40 - Наиболее распространенные уязвимости внутренней сети (доля систем)

Также в качестве рекомендаций предлагается отключить неиспользуемые в локальной вычислительной сети протоколы канального и сетевого уровня. Если эти протоколы требуются для работы каких-либо систем, следует выделить для них отдельный сегмент сети, к которому нет доступа из пользовательского сегмента.

КИС остаются уязвимыми для атак злоумышленников. С каждым годом увеличивается доля компаний, где удается получить доступ к ресурсам внутренней сети от лица внешнего злоумышленника. Использование методов социальной инженерии и эксплуатация недостатков защиты беспроводных сетей дополнительно повышают шансы на успешное преодоление сетевого периметра. Как правило, векторы атак основываются

на эксплуатации известных недостатков безопасности и в большинстве своем не требуют от злоумышленника глубоких технических знаний.

Для поддержания высокого уровня защищенности системы необходимо соблюдать общие принципы и рекомендации обеспечения информационной безопасности. Необходимо регулярно проводить анализ защищенности веб-приложений, при этом наиболее эффективным методом проверки является метод белого ящика, подразумевающий анализ исходного кода. В качестве превентивной меры рекомендуется использовать межсетевой экран уровня приложений (web application firewall) для предотвращения эксплуатации уязвимостей, которые могут появляться при внесении изменений в код или добавлении новых функций.

В рамках тестирования на проникновение действия экспертов редко обнаруживаются службой безопасности компаний, следовательно, и реальные злоумышленники могут долгое время находиться в инфраструктуре и оставаться незамеченными. Поэтому важно не только защищать сетевой периметр, но и проводить регулярный ретроспективный анализ сети с целью выявить уже случившееся проникновение. Для поиска следов компрометации рекомендуется использовать специализированные средства глубокого анализа сетевого трафика, способные обнаруживать сложные целевые атаки как в реальном времени, так и в сохраненных копиях трафика. Такое решение даст возможность не только увидеть факты взлома, но и отслеживать сетевые атаки, в том числе запуск вредоносных утилит, эксплуатацию уязвимостей КИС и атаки на контроллер домена. Это позволит уменьшить время скрытного присутствия нарушителя в инфраструктуре и тем самым минимизировать риски утечки важных данных и нарушения работы бизнес-систем, снизить возможные финансовые потери. Дополнительно рекомендуется использовать специальное антивирусное ПО, которое проверяет файлы в изолированной среде,

выявляет присутствие вредоносных программ и помогает блокировать вредоносную активность.

Важно следовать всем рекомендациям в комплексе, так как даже отдельные пробелы в механизмах защиты могут послужить причиной взлома инфраструктуры и компрометации критически важных ресурсов. Рекомендуется регулярно проводить тестирование на проникновение, чтобы выявлять векторы атак на корпоративную систему и на практике оценивать эффективность принятых мер защиты.

ГЛАВА 4 АНАЛИЗ ЭФФЕКТИВНОСТИ ПРИМЕНЕНИЯ КОМПЛЕКСНОГО ТЕСТИРОВАНИЯ КОРПОРАТИВНЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

4.1 Оценка осведомленности об уровне уязвимости корпоративных информационных систем

В дополнение к работам по тестированию на проникновение важно проводить проверки осведомленности сотрудников в вопросах информационной безопасности. Работы выполняются по заранее согласованным сценариям, которые имитируют реальную атаку злоумышленника. Проверки осуществляются путем телефонного взаимодействия и рассылки электронных писем. В телефонном разговоре предпринимаются попытки узнать у пользователей ту или иную ценную информацию. Электронные письма содержат вложенные файлы или ссылку на веб-ресурс, где требуется ввести учетные данные. В ходе проверки фиксируется реакция сотрудников: факты перехода по ссылке, ввода учетных данных или запуска вложения.

По статистике почти треть пользователей переходит по ссылке или запускает приложенный файл, а каждый десятый сотрудник вводит свои учетные данные в фальшивую форму аутентификации. Заметная доля пользователей (14%) раскрывает конфиденциальную информацию в разговоре по телефону или вступает в переписку с условным злоумышленником, сообщив при этом дополнительные сведения о компании: имена и должности сотрудников, номера внутренних и мобильных телефонов (рис. 4.1).

С целью выявления и предотвращения атак методами социальной инженерии рекомендуется регулярно проводить обучение сотрудников, направленное на повышение их компетенции в вопросах информационной безопасности, с контролем результатов.

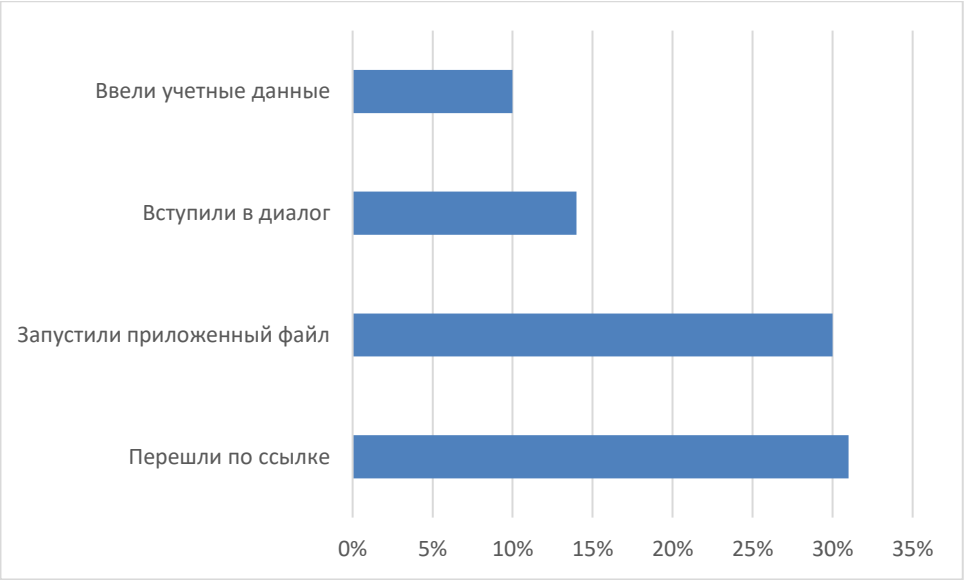


Рисунок 4.1 - Результаты оценки осведомленности сотрудников

Беспроводные сети являются потенциальным вектором проникновения во внутреннюю инфраструктуру компании. Злоумышленнику достаточно установить на ноутбук общедоступное ПО для атак на беспроводные сети и приобрести недорогой модем, который может работать в режиме мониторинга трафика. В семи из восьми протестированных систем беспроводные сети доступны за пределами контролируемой зоны, а значит, злоумышленник может проводить атаки, просто находясь на близлежащей территории, например на парковке рядом с офисом или в кафе на цокольном этаже здания. Почти во всех сетях исследуемых компаний используется протокол WPA2 с методами аутентификации PSK или EAP.

В зависимости от используемого метода аутентификации проверяется возможность реализации разных типов атак. Для WPA2/PSK проводится перехват рукопожатия между точкой доступа и легитимным клиентом точки доступа с последующим подбором паролей методом перебора. Успех этой атаки обусловлен сложностью пароля. В ходе проверок было установлено, что словарные ключи для подключения к беспроводной сети

используются в половине систем. Другой способ атаки - создание поддельной точки доступа - применяется для любого метода аутентификации. Если при подключении к беспроводной сети не производится проверка подлинности сертификатов, то злоумышленник может создать поддельную точку доступа с идентичным названием сети (ESSID) и более мощным сигналом, чем у оригинальной. В случае подключения клиента к этой точке доступа злоумышленник получает его идентификатор в открытом виде и значение NetNTLM v1 challenge-response, с помощью которого может подобрать пароль методом перебора.

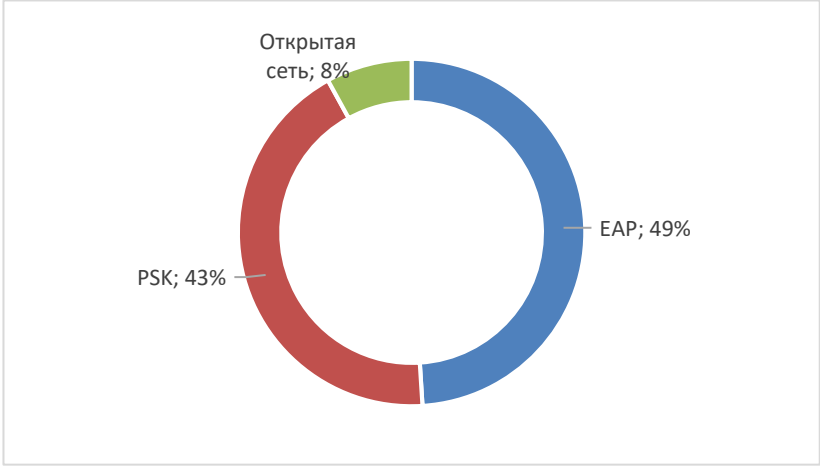


Рисунок 4.2 - Методы аутентификации в беспроводных сетях

Результат атаки зависит от того, насколько сотрудники компании осведомлены в вопросах информационной безопасности.

Рекомендуется регулярно проводить анализ защищенности беспроводных сетей, чтобы выявлять ошибки конфигурации и потенциальные векторы проникновения во внутреннюю сеть.

4.2 Анализ эффективности комплексного тестирования КИС данными

Перед любым интеграционным процессом всегда стояла задача улучшения работы комплекса информационных систем.

В качестве критериев оценки хорошо построенного интеграционного процесса можно выделить следующие:

- скорость передачи данных;
- эффективность обмена данными;
- оценка пользователей интеграционных систем.

На рисунке 4.3 представлена гистограмма, отображающая разницу в скорости работы между интеграционными операциями до и после апробации сценария тестирования.

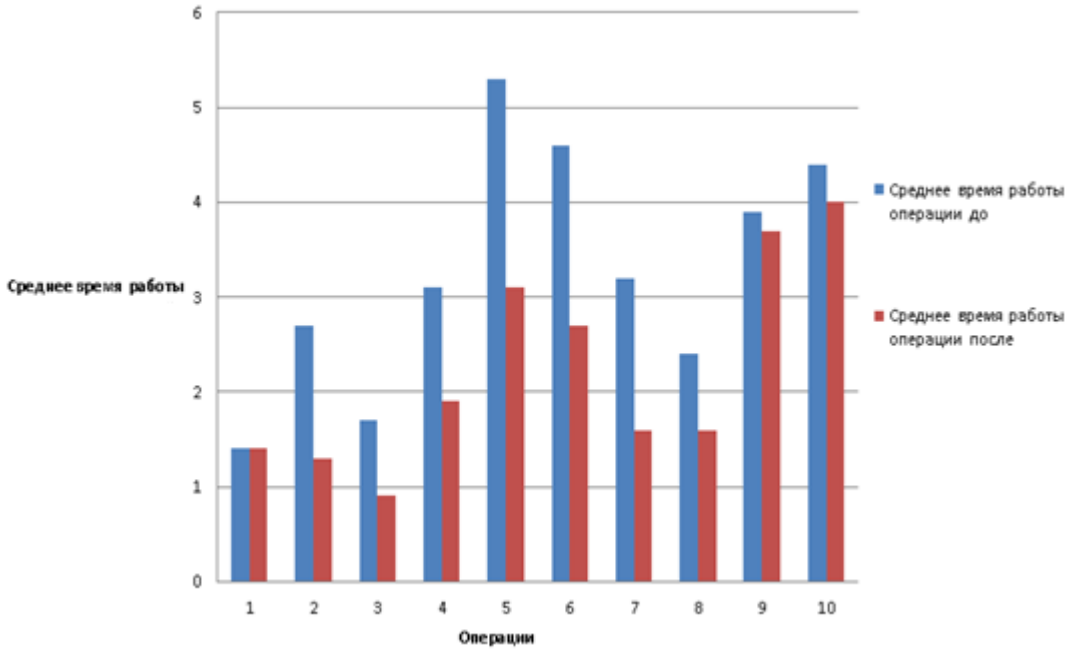


Рисунок 4.3 - Среднее время работы интеграционных операций

По данным гистограммы можно сделать вывод, что предложенный сценарий тестирования, включающий методы комплексного тестирования, способствует повышению качества работы КИС, но для чистоты эксперимента, чтобы оценить достоверность результатов исследования, необходимо воспользоваться критериями эффективности.

Выдвинем основную (нулевую) гипотезу H_0 и проверим, не противоречит ли она имеющимся эмпирическим данным. Конкурирующей (альтернативной) гипотезой назначим H_1, которая противоречит нулевой.

Рассмотрим использование t-критерия Стьюдента для определения наличия различий между двумя выборками. При этом выборки могут быть:

- независимыми, несвязными с разным числом значений в выборках;
- зависимыми, связанными с равным числом значений в выборках.

Критерий t Стьюдента направлен на оценку различий величин средних $\overline{x}$ и $\overline{y}$ двух выборок X и Y, которые распределены по нормальному закону. Одним из главных достоинств критерия является широта его применения.

Как правило, в исследованиях чаще всего применяются такие параметрические критерии как: t-критерий Стьюдента, позволяющий оценивать различия средних в двух выборках и F - критерий Фишера, который оценивает различия между двумя дисперсиями. Они позволяют получить наиболее наглядное представление анализируемых совокупностей. Вычисление значения $t_{эмп}$ осуществляется по формуле:

$$t_{эмп} = \frac{d}{Sd} \tag{4.1}$$

$$\text{где } d = \frac{\sum d_i}{n} = \frac{\sum (x_i - y_i)}{n} \tag{4.2}$$

где

$d_i = x_i - y_i$ – разности между соответствующими значениями переменной X и переменной Y, d среднее этих разностей, n – объем выборки, пределы суммирования от i = 0 до n.

В свою очередь Sd вычисляется по следующей формуле:

$$Sd = \sqrt{\frac{\sum d_i^2 - \frac{(\sum d_i)^2}{n}}{n*(n-1)}} \tag{4.3}$$

Число степеней свободы определяется по формуле k = n - 1.

Для применения t-критерия Стьюдента необходимо соблюдать следующие условия:

1. Измерение может быть проведено в шкале интервалов и отношений.
2. Сравниваемые выборки должны быть распределены по нормальному закону.

Для расчета t-критерия Стьюдента необходимо воспользоваться процедурой «Парный двухвыборочный t-тест для средних» из пакета анализа программного продукта Microsoft Excel (рис. 4.4).

Нулевая гипотеза $H_0: \mu_1 - \mu_2 = \delta$ принимается, если $|t| < t_{кр1}$ (в противном случае отвергается); гипотеза H_0 при конкурирующей гипотезе $H_1: \mu_1 > \mu_2 + \delta$ принимается, если $t < t_{кр2}$; при конкурирующей гипотезе $H_1: \mu_1 < \mu_2 + \delta$ нулевая гипотеза принимается при выполнении неравенства $t_{кр2} < t$.

	Переменная 1	*Переменная 2*
Среднее	3,27	2,22
Дисперсия	1,626777778	1,166222222
Наблюдения	10	10
Корреляция Пирсона	0,817653538	
Гипотетическая разность средних	0	
df	9	
t-статистика	4,516158041	
P(T<=t) одностороннее	0,000727396	
t критическое одностороннее	1,833112933	
P(T<=t) двухстороннее	0,001454792	
t критическое двухстороннее	2,262157163	

Рисунок 4.4 - Результат расчета процедуры «Парный двухвыборочный t-тест для средних»

С помощью процедуры «Парный двухвыборочный t-тест для средних», проверим на уровне значимости $\alpha = 0.05$ гипотезу H_1: средние временные интервалы процессов обмена данными между системами с применением предложенной методики комплексного тестирования существенно уменьшились. Основная гипотеза H_0: среднее время процессов обмена данными между системами без применения предложенной методики и с ее применением существенно не изменилось.

Анализ результатов решения показал, что расчетное значение $t = 4,5$ статистики T находится в области $(2,26; +\infty)$. Это означает, что гипотеза H_0 о равенстве времени работы интеграционных операций без применения

предложенной методики и с ее применением противоречит фактическим данным наблюдения и, следовательно, ее надо отклонить (на уровне значимости $\alpha = 0.05$) и принять альтернативную гипотезу H_1 предполагающую, что среднее время процессов обмена данными между модулями КИС без применения предложенной методики тестирования больше времени работы после внедрения.

Таким образом, первоначальное предположение подтвердилось, действительно, среднее время работы интеграционных операций, после внедрения сценария комплексного тестирования КИС дает лучшее результаты по сравнению с предыдущими. На основании статистической оценки интеграционного эксперимента можно констатировать, что проделанная работа достаточно значима и обоснована.

ЗАКЛЮЧЕНИЕ

В ходе проведенного исследования были рассмотрены наиболее популярные методы тестирования, применимые для проведения комплексного тестирования корпоративных информационных систем. Показано, что комплексное тестирование КИС играет важную роль в процессе жизненного цикла ИС для определения ее работоспособности и надежности.

При проведении исследований в работе получены следующие теоретические и прикладные результаты.

1. В ходе анализа теоретических положений жизненного цикла тестирования КИС были получены концепции, необходимые для определения понятия комплексного тестирования и выделения его основных видов.

2. Для разработки тестов подробно рассмотрены методы функциональных диаграмм и попарного тестирования. Показаны преимущества проведения анализа предметной области на основе онтологической модели.

3. На основе сравнительного анализа методов тестирования сделаны выводы, что для более качественного проведения комплектного тестирования КИС необходимо использование совокупности рассмотренных методов. Для предложенных методов определены основные концепции и этапы проведения тестирования.

4. В рамках нагрузочного тестирования рассмотрен подход, основанный на моделях. Показано его эффективное применение для тестирования прикладного программного обеспечения в IT-среде функционирования, включающей СУБД, базы данных, системное ПО, развернутые на оборудовании комплекса технических средств ИС.

5. Подробно исследована технология нагрузочного тестирования с использованием моделей, сформированы требования к проведению нагрузочного эксперимента и свойства объекта тестирования, охватывающие все аспекты планирования, проведения нагрузочного эксперимента и анализа его результатов. Показано, что использование моделей существенно снижает

трудоемкость нагрузочного тестирования и обеспечивает возможность повторного использования подготовленного эксперимента.

6. Рассмотрен ряд инструментальных средств, предоставляющих возможности для реализации автоматизированного тестирования. Каждый из рассмотренных инструментов был оценен с точки зрения его внедрения в существующий процесс разработки, а также с точки зрения удовлетворения требованиям КИС. Приведены результаты анализа по основным из предложенных инструментов. Показано, что автоматизация процесса тестирования помогает компаниям сокращать время, затрачиваемое на тестирование, а также упрощать весь процесс, так как применяются специальные программные инструменты для создания и запуска тестов, а также проверки результатов их выполнения.

7. С целью выявления недостатков защиты различных компонентов Системы и определения потенциальных атак на информационные ресурсы, рассмотрены наиболее эффективные способы анализа защищенности ИС. Даны ряд рекомендаций по повышению защищенности ИС и информационной безопасности. Показано, что для поддержания высокого уровня защищенности системы необходимо соблюдать общие принципы и рекомендации обеспечения информационной безопасности. Необходимо регулярно проводить анализ защищенности веб-приложений, при этом наиболее эффективным методом проверки является метод белого ящика.

Таким образом, предложенную методику (сценарий) комплексного тестирования КИС можно использовать для оценки надежности и эффективности ИС, функционирующей в бизнес-среде.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Нормативно-правовые акты

1. ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения (ИСО 5807-85). Введ. 1992-01-01. – М.: Издательство стандартов, 1992 – 14 с. – (Единая система программной документации).

2. ГОСТ Р ИСО/МЭК 12207-99. Информационная технология. Процессы жизненного цикла программных средств. Введ. 2000-07-01. – М.: Издательство стандартов, 2000. – 30 с.

3. ГОСТ 7.82-2001. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления. – Введ. 2002-07-01. – Минск: Издательство стандартов, 2001. – 35 с. – (Система стандартов по информации, библиотечному и издательскому делу).

4. ГОСТ Р ИСО/МЭК 9126-93 “Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению”

Научные издания

5. Дастин Э. Тестирование программного обеспечения. Внедрение, управление и автоматизация / Э. Дастин, Д. Рэшка, Д. Пол; Пер. с англ. М. Павлов. - М.: Лори, 2013. - 567 с.

6. Козлов А.Н., Позин Б.А. Технология комплексного нагрузочного тестирования банковской системы, международная научно-практическая конференция «Реинжиниринг бизнес-процессов на основе современных информационных технологий системы управления знаниями», 2006 г. (РБП-СУЗ-2006), Москва. С. 130-131.

7. Котляров В.П. Основы современного тестирования программного обеспечения: учебное пособие/В.П.Котляров, Т.В.Коликова – СПб.: Питер, 2004. – 170 с.

8. Куликов С. С. Тестирование программного обеспечения. Базовый курс : практ. пособие. / С. С. Куликов. – Минск: Четыре четверти, 2015. – 294 с.

9. Липаев В.В. Тестирование крупных комплексов программ на соответствие требованиям: учебник/В.В.Липаев – М.: ИПЦ «Глобус», 2008. – 316 с.

10. Липаев В.В., Позин Б.А., Блау С.А., Анализ стратегий тестирования логики программ, Кибернетика, 1982, "2, с 45-50.

11. Певченко С. С. Методы анализа данных // Молодой ученый. — 2015. — №13. — С. 167-169.

12. Савин Р. Тестирование Дот Ком, или пособие по жестокому обращению с багами в интернет-стартапах/Р.Савин – М.: Дело, 2007. – 312 с.

13. Савин Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах. – М.: Дело, 2007. – 312 с

14. Сергеева Н.А. Стратегия тестирования информационных систем управления вузом на основе документов с теговой разметкой // Вестник РУДН, серия Информатизация образования. - 2012. - № 3. - С. 31-37.

15. Степанченко И. В. Эквивалентное разбиение. Методы тестирования программного обеспечения. Волгоград : РПК "Политехник", 2006.

16. Сэм Канер. Testing Computer Software/Сэм Канер, Джек Фолк. – М.:ДиаСофт, 2001. – 544 с.

17. Филиппов В. А., Хатько Е. Е. Генерация тестовых сценариев для мобильных приложений. Информационные, сетевые и телекомуникационные технологии. 2012.Т. 4.

18. Филиппов В. А., Хатько Е. Е. Проблемные вопросы автоматизации тестирования для мультизадачных пользовательских комплексов. Информационные, сетевые и телекомуникационные технологии. 2012 . Т. 4.

19. Хатько Е. Е. Москва, Долгопрудный : МФТИ, 2009. Один из подходов к анализу системы тестирования сложных программных комплексов. Современные проблемы фундаментальных и прикладных наук. Т. 1, с. 104 -107.

20. Хатько Е. Е. Об одном методе тестирования «мобильных» приложений. Труды МФТИ. 2012

21. Хатько Е. Е. Современные проблемы фундаментальных и прикладных наук. Один способ реализации алгоритма генерации тестов в тестировании на основе моделей. Т. 1, с . 92-95. 52. М. 2010.

22. Хатько Е. Е., Филиппов, В. А. Проблемы качества тестирования программного обеспечения для мультизадачных пользовательских комплексов. Качество. Инновации. Образование. 3 2011.Т. 3, с. 32-35.

23. Шмейлин Б. 3. Современные технологии тестирования WEB приложений. Системы и средства информатики. 2009 г., с. 138-147.

Электронные ресурсы

24. Бородин А.М., Мирвода С.Г., Поршнев С.В. Особенности тестирования устойчивости к сбоям корпоративных информационных систем методом генерирования отказов // Современные проблемы науки и образования. – 2014. – № 5. URL: http://www.science-education.ru/ru/article/view?id=14997 (дата обращения: 07.03.2019).

25. Волков В.Г., Автоматизированная система контроля и обеспечения надежности программных средств // http://www.unn.ru/pages/issues/vestnik/99999999_West_2009_5/27.pdf (дата обращения 25.02.2019)

26. Ключёв А.О., Маковецкая Н.А., Проблемы тестирования системного информационного обеспечения распределённых информационно-управляющих систем // http://www.ict.edu.ru/ft/001795/vestnik10_7.pdf

27. Котов С.Л., Нагрузочное тестирование как элемент формирования безопасных систем // http://www.ooogic.ru/downloads/loading%20test%20as%20an%20element.pdf

28. Кулямин В. В. Тестирование на основе моделей, URL: http://panda.ispras.ru/~kuliamin/lectures-mbt/Lecture04.pdf. (дата обращения 05.02. 2019).

29. Петренко А. Тестирование на основе моделей. Информационные системы. URL: http://www.info-system.ru/testing/testtestingbasismodels.html. (дата обращения 21.02. 2019)

30. Старолетов С.М., Крючкова Е.Н. Тестирование распределенных приложений на основе построения моделей [Электронный ресурс]: учебное пособие/С.М.Старолетов, Е.Н.Крючкова. – Режим доступа: http://cyberleninka.ru/article/n/testirovanie-raspredelennyhprilozheniy-na-osnove-postroeniya-modeley

31. Технология каскадного тестирования программного обеспечения. – Режим доступа: http://software-testing.ru/about/trainers/94-rukol

32. Уровни тестирования программного обеспечения. – Режим доступа: http://www.protesting.ru/testing/testlevels.html

Литература на иностранном языке

33. Andrews A., Offut J., Alexander R. Testing Web Applications by Modeling with FSMs. б.м. : National Science Foundation, 2005.

34. Changyou Xing, Guomin Zhang, Ming Chen. Research on universal network performance testing model/ International Symposium on Communications and Information Technologies, 2007. P. 780-784.

35. Heiskanen H., Maunumaa M., Katara, M. Test Process Improvement for Automated Test Generation. Tampere: Tampere University of Technology, Department of Software Systems, 2010.

36. Jie M., Honlin Zh., Wenbo X., Jin L., Reliability Testing Methods for Critical Information System based on State Random // http://www.ipcsit.com/vol16/6-ICICM2011M009.pdf

37. Khatko E, Fillipov V. Mobile applications testing processes metrics and optimization criteria. Software Engineering. 5, 2012 г., Т. 2.

38. Kim G.-B. F method of generating massive virtual clients and model-based performance test/ Fifth International Conference on Quality Software, 2005. P. 250-254.

39. Kostogryzov A., V.Panov, B.Pozin, V.Sablin. Mathematical modeling of processes in systems life cycles in compliance with standarts requirements of ISO/IEC 15288 and ISO/IEC 12207, Spincose, Montreal, Canada, 2003.

40. Makinen M. Model Based Approach to Software. Helsinki : Helsinki University of Technology, 2007.

41. MSDN Magazine, microsoft.com. URL: http://msdn.microsoft.com/ra-ru/magazine/dd419663 .aspx. (Retrieved. 09.11.2018)

42. Robinson H. Graph Theory Techniques in Model-Based Testing. 1999.

43. The Next Generation 1996 Lexicon A to Z". Next Generation. No. 15. Imagine Media. Alpha software generally barely runs and is missing major features like gameplay and complete levels. March 1996. p. 29.

44. Xijiang L., Pomeranz I., Sudhakar M. Techniques for Improving the Efficiency of Sequential Circuit Test Generation. Iowa : University of Iowa. 2015.