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)

09.04.03 Прикладная информатика
(код и наименование направления подготовки)

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

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему Разработка методики тестирования программного обеспечения

Студент Ю.А. Серякова _____
(И.О. Фамилия) (личная подпись)

Научный руководитель к.п.н., доцент, О.М. Гущина
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

ОГЛАВЛЕНИЕ

Введение

Основной всплеск интереса к теме тестирования пришёлся на конец 20-го века. Бурное развитие различных технологий привело к росту рынка производства программного обеспечения и к пересмотру вопросов обеспечения качества и надёжности разрабатываемых программ. Резко усилившаяся конкуренция между производителями ПО потребовала особого внимания к качеству создаваемых продуктов, т.к. теперь у потребителя был выбор: многие фирмы предлагали свои продукты и услуги по достаточно приемлемым ценам, а потому можно было обратиться к тем, кто разработает программу не только быстро и дёшево, но и качественно. Ситуация осложнилась тем фактом, что в настоящее время практически все сферы человеческой жизни подвержены компьютеризации, тем самым вопрос о качестве ПО начинает приобретать особую важность, так как сегодня это уже не просто комфорт от работы в той или иной программе, сегодня ПО управляет оборудованием в больницах, диспетчерскими системами в аэропортах, атомными реакторами, космическими кораблями и т.п.

Сегодня тестирование стало обязательной частью процесса производства ПО. Оно направлено на обнаружение и устранение как можно большего числа ошибок. Следствием такой деятельности является повышение качества ПО по всем его характеристикам.

Однако, несмотря на существование большого количества разных методов и видов тестирования, не существует единой методики для проверки ПО, вся информация о том, как нужно тестировать и что для этого предпринять разрозненна по разным научным источникам.

Анализ актуальности обусловили выбор **темы** исследования: «Разработка методики тестирования программного обеспечения».

Научной проблемой в данной работе является отсутствие единой методики тестирования.

Гипотеза исследования состоит в предположении, что применение разработанной методики тестирования в рамках данного диссертационного исследования поможет улучшит качество выдаваемого функционала.

Целью исследования является разработка методики тестирования программного обеспечения и ее применение на примере.

Для достижения поставленной цели необходимо решить следующие **задачи**:

1) выявить место тестирования в процессе разработки ПО;
2) рассмотреть существующие этапы тестирования;
3) дать определение термину «методика тестирования»;
4) провести анализ существующих видов и методов тестирования;
5) принять решение, какие виды и методы тестирования должны быть покрыты новой методикой;
6) предоставить новую методику тестирования;
7) протестировать новую методику на реальном примере;
8) описать возможности применения новой методики тестирования;
9) обосновать эффективность новой методики тестирования;
10) определить дальнейшее направление в развитии данной методики.

Объектом исследования является процесс тестирования ПО.

Предметом исследования является методика тестирования ПО.

Методы исследования включают в себя:

1) методы тестирования программного обеспечения;
2) методы определения качества продукта;
3) методы верификации программного обеспечения;
4) классификацию;
5) систематизацию;
6) сравнительный анализ.

Теоретической основной исследования стали:

1) отечественные и зарубежные исследования по обеспечению качества программного обеспечения;

2) публикации на сайтах, посвященные тестированию программного обеспечения.

Теоретическая значимость работы заключается в том, что в ней проанализированы большинство существующих видов и методов тестирования и систематизированы в одну методику.

Практическая значимость работы заключается в применении новой методики на реальном проекте в компании ООО «НетКрэкер».

Научная новизна работы состоит в разработке новой методики тестирования, которая поможет начинающим и практикующим тестировщикам улучшить качество своего тестирования.

На защиту выносятся:

1) Разработанная методика тестирования с ее подробным описанием и делением на конкретные шаги;
2) Анализ эффективности разработанной методики.

Магистерская диссертация состоит из введения, четырех глав и заключения, изложенных на 91 страницах, а также списка использованной литературы (47 наименований) и 4 приложений.

Глава 1 Теоретико-методологические основы тестирования программного обеспечения

1.1 Понятие тестирования и его место в процессе разработки ПО

Понятие тестирования рассматривается как в учебниках, так и в нормативных актах. В результате анализа различных источников было выявлено, что наиболее подходящее определение дано в стандарте IEEE Std 829-1983: «тестирование – это процесс анализа программного обеспечения, направленный на то, чтобы выявить, насколько его реально существующие свойства соответствуют требуемым свойствам, другими словами, выявить дефект и оценить свойства ПО» [2].

В ГОСТ Р ИСО МЭК 12207-99 жизненный цикл ПО содержит такие вспомогательные процессы, как аттестация, верификация, совместный анализ и аудит. В процессе верификации определяется то, что программные продукты работают согласно требованиям или условиям, реализованным в предыдущих работах. В данный процесс может быть включен анализ, проверка и испытание (тестирование). Процесс аттестации представляет собой процесс определения того, насколько по́лно установленные требования, созданная система или программный продукт соответствуют их функциональному назначению. В процессе совместного анализа оцениваются состояния и, если нужно, результаты работ (продуктов) по проекту. В процессе аудита определяется соответствие требованиям, планам и условиям договора. Все рассмотренные процессы в совокупности и составляют то, что обычно называется тестированием.

Согласно мнению многих авторов [10, 11, 17, 23-27] основу тестирования составляют тестовые процедуры с определенными входными данными, начальными условиями и ожидаемым результатом, которые разработаны для конкретной цели, например, проверки отдельной программы или верификации соответствия на определенное требование. Тестовые процедуры могут использоваться для различных аспектов функционирования программы – от

правильной работы конкретной функции до адекватного выполнения бизнес-требований.

В научно-популярном издании Криспин Л. и Грегори Д. «Гибкое тестирование: практическое руководство для тестировщиков программного обеспечения и гибких команд» рассмотрен гибкий подход к тестированию. В современных IT-компаниях все чаще и чаще применяется именно гибкий подход к разработке, а, следовательно, и к тестированию. В книге рассмотрена роль тестировщика в гибком проекте, как взаимодействовать с разработчиками, сколько и как нужно автоматизировать.

В работе подробно рассмотрены виды тестирования, роли тестировщиков в нем, приемы и технологии по определению законченности тестирования [22].

Для выявления места тестирования в процессе разработки ПО были рассмотрены и проанализированы функции отделов стандартной фирмы-разработчика ПО. После анализа был сделан вывод, что в процессе разработки ПО все отделы взаимодействуют друг с другом, и тестировщики играют свою роль не только на этапе тестирования, но и на других. Например, в IT-компаниях практически всегда существует аналитический отдел, который помимо определения концепции и направления развития программного продукта, может заниматься определением функциональных возможностей системы, описанием бизнес-процессов, а также иногда отслеживать изменения в программном продукте и вносить их в документацию. Всем этим также могут заниматься и тестировщики (особенно часто это встречается в гибких методологиях) [25].

Одним из ключевых отделов, без которых невозможно обойтись в IT-фирмах, является, конечно же, отдел разработки. Помимо реализации запланированных функций программы, разработчики часто проводят unit-тестирование, находят недочеты в коде и исправляют их, а также осуществляет сборку и выпуск программного обеспечения. Во всем этом (помимо написания кода) тестировщики могут помогать отделу разработки, в частности, например,

во многих компаниях, отдел разработки не занимается выпуском сборки, и отдел тестирования полностью забирает данную функцию на себя [46].

Помимо прочих, в компаниях очевидно присутствует отдел тестирования, и, проанализировав разные источники, можно сделать вывод, что у него существуют следующие задачи:

- осуществление комплексного контроля качества;
- подготовка тестовой документации (часто первые варианты создаются на стадии обсуждения проекта, до начала разработки, т.е. совместно с аналитическим отделом);
- нахождение и исправление ошибок в работе программных продуктов (совместно с отделом разработки);
- проверка соответствия документации программного продукта стандартам и реально разработанным функциям (совместно с аналитическим отделом и отделом разработки);
- участвует в разработке и внедрении системы качества;
- автоматизирует тестирование (совместно с отделом разработки);
- оценивает производительность разрабатываемых программных средств на различных программно-аппаратных платформах и их специфических конфигурациях [13].

Таким образом, тестирование – это процесс, направленный на то, чтобы выявить, насколько реализованные свойства разрабатываемого ПО соответствуют требуемым. Помимо этого, тестирование применяется на разных этапах проекта и с помощью него проверяются не только определённые функции программы, но и проверяются адекватность бизнес-требований, описываются различные бизнес-процессы вместе с отделом аналитики, также иногда тестировщики включены в процесс сборки и выпуска программного обеспечения вместе с отделом разработки.

1.2 Этапы тестирования

Тестирование ПО в различных случаях может осуществляться по-разному, но в целом проводится в несколько этапов. Это планирование тестирования и анализ требований, разработка тестовых сценариев, выполнение тестов, анализ результатов тестирования и выдача проекта заказчику [30].

1.2.1 Планирование тестирования и анализ требований

Существует множество работ с описанием различных стадий тестирования. Проанализировав некоторые из них, можно сделать вывод, что некоторые авторы забывают о таком важном шаге как анализ требований, хотя он должен являться неотъемлемой частью любого IT-проекта и тестировщики должны принимать в нем участие и взглянуть на требования со стороны будущего тестирования.

В общих чертах этап планирования – это организация работы, согласование тематики проекта с заказчиком, выработка идей проекта, сбор и анализ требований, определение функциональных характеристик продукта. Результатом этапа являются план реализации проекта и техническое задание [25].

В ходе планирования и анализа требований «тестируются» идеи. На данном этапе к анализу могут быть привлечены специалисты и руководители разных направлений, в том числе отдела маркетинга, отдела разработки и др.

Специалисты по тестированию знакомятся с проектными документами и собирают различного рода информацию, которая может оказать помощь в оценке результатов этапа и дальнейшем планировании тестирования. Для сбора информации используются следующие способы:

- сравнительный анализ: выполняется сравнение целей и задач проекта с аналогичными или похожими проектами для установления взаимосвязей и выявления потенциальных проблем;
- дискуссионные группы: выполняется анализ и обсуждение предлагаемых идей с целью уточнения и детализации;

– обследование объекта: выполняется изучение бизнес процессов с целью выявления скрытой информации.

Логическим завершением каждой из этих процедур является пересмотр существующих планов.

Для анализа требований используются различные методы, основными из них являются интервью, мозговой штурм, анкетирование, наблюдение, работа с фокусными группами, моделирование процессов и взаимодействий и т.п.

Форма представления, степень детализации и перечень полезных свойств требований зависят от уровней и типов требований, которые схематично представлены на рисунке 1.1 [30].

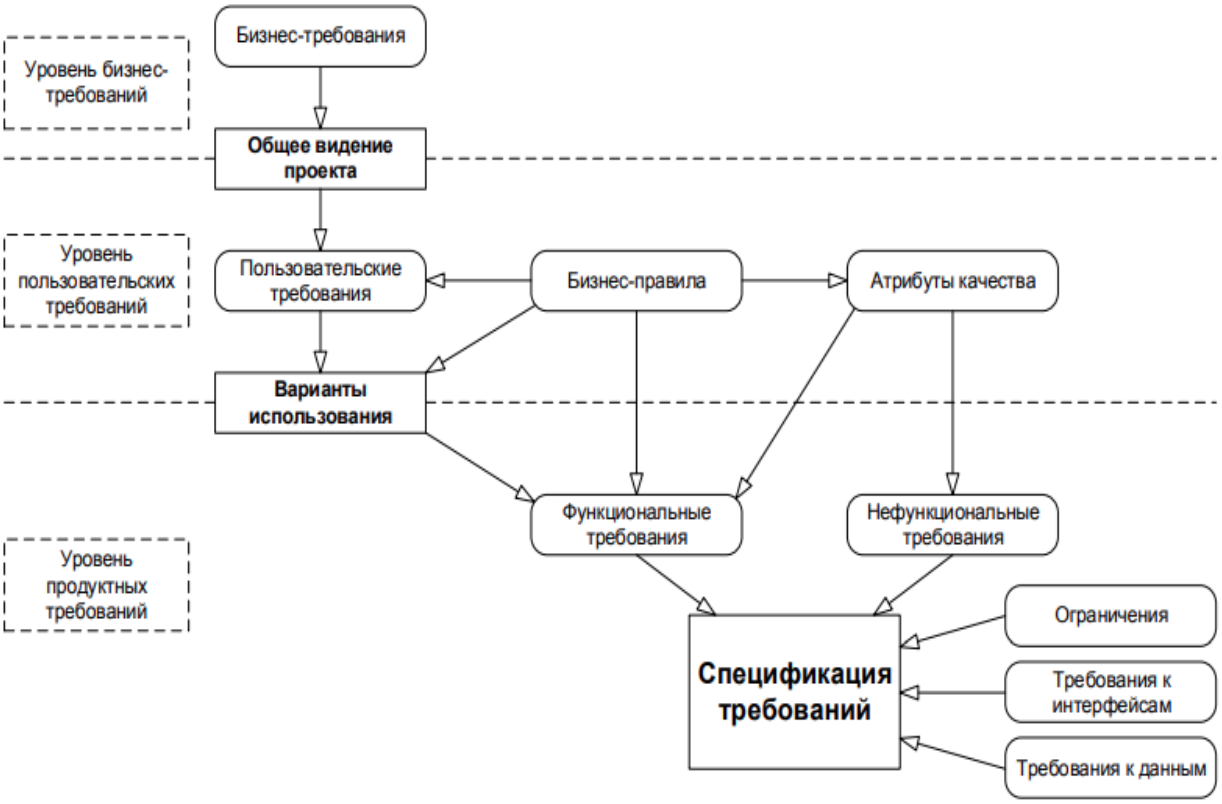


Рисунок 1.1 – Уровни и типы требований

По рисунку видно, что требования бывают трех уровней: уровень бизнес-требований, уровень требований пользователей и уровень требований к продукту.

Бизнес-требования – это цель, которая ставится при разработке приложения. В результате получается общее видение проекта.

Требования пользователей описывают задачи, для решения которых создается система. Пользовательские требования могут быть представлены в виде вариантов использования, пользовательских историй, пользовательских сценариев.

Требования к продукту наиболее важны при проведении тестирования. Это требования к выполняемым функциям, нефункциональные требования (удобство, надежность, безопасность и т.д.), ограничения (технические характеристики ПК, ресурсы и т.д.), требования к интерфейсу, требования к данным [47].

Проанализировав различные источники, можно выделить основные цели, которые должны быть достигнуты в ходе анализа требований:

- адекватность требований. В ходе анализа требований может выясниться, что заказчик хочет получить совершенно другой продукт. Группа тестирования должна удостовериться, что заявленные требования соответствуют ожиданиям заказчика.
- полнота требований. Выяснение дополнительной информации у заказчика на последующих этапах проекта может привести к дополнительным задержкам. Кроме этого, невыясненные детали могут привести к кардинальному усложнению проекта.
- совместимость требований. Функции разрабатываемого программного обеспечения могут оказаться несовместимыми по логическим (противоречивость функций) или психологическим (концептуальные различия) компонентам.
- выполнимость требований. Группа тестирования должна выяснить возможность нормальной эксплуатации продукта. Заявленные требования могут подразумевать более высокие требования к аппаратному обеспечению, памяти, пропускной способности каналов связи и т.д.
- разумность требований. Качество продукта (его производительность, надежность и нетребовательность к ресурсам) и стоимость и сроки его разработки являются двумя противоречивыми требованиями. Поэтому

расстановка приоритетов при планировании является ключевой составляющей конечного успешного продукта.

– подверженность тестированию. Позволяет определить соответствие документации и требований.

Таким образом, этап анализа требований является важнейшим этапом тестирования, так как на этом этапе выдвигаются требования к функциям будущего ПО, и руководство IT-проектов не должно забывать о важной роли тестировщиков в данном процессе, так как они могут выявить соответствуют ли заявленные требования ожиданиям заказчика, удостовериться в полноте требований, расставить приоритеты при планировании будущих работ, а также сопоставить составленную документацию и требования заказчика, тем самым избежав уточнений по документации на последующих этапах проекта.

1.2.2 Разработка тестовых сценариев

В ходе выполнения данного этапа разработчики создают код и программную документацию. Задачей группы тестирования является разработка тестовых сценариев. Существует множество определений тому, что же такое тестовый сценарий, однако большинство теоретиков склоняются к тому, что тестовый сценарий – это «тестовая документация, в которой записаны стандартные и альтернативные сценарии работы с приложением, используемые при тестировании очередной версии приложения» [31] или «описание последовательности действий в системе и ожидаемого поведения» [32].

Тестовые сценарии помогают тестировщикам при проведении различных видов тестирования, например, функционального тестирования, приемочного, нагрузочного, исследовательского и пр.

Для разработки тестовых сценариев и последующего выполнения тестов часто используют системы управления тестированием, которые повышают производительность тест-дизайнеров и тестировщиков, а также обеспечивают прозрачность уровня качества разрабатываемой программы среди всех участников [29].

Помимо этого, не стоит забывать, что тестовые сценарии должны отражать технические требования к программе, в случае изменения которых, соответственно должны быть отредактированы составленные сценарии.

Также стоит отметить, что тестовые сценарии удобно объединять в различные группы. Разные авторы классифицируют такие группы по-разному, но в данной работе мы склоняемся к объединению тестовых сценариев по назначению, например, тестирование версии продукта, тестирование удобства использования, тестирование безопасности, тестирование конфигурации, и, конечно же, не стоит забывать о тестировании основного функционала, которое также можно разделить на различные группы, но данные группы будут зависеть от конкретного проекта. Самым элементарным разделением считается позитивные сценарии и негативные.

Помимо этого, тестовые сценарии также можно «протестировать», а точнее у составленных тестовых сценариев есть определенные характеристики, которым они должны соответствовать:

- тестовый сценарий с высокой вероятностью может выявить ошибки в основном функционале;
- набор тестов не избыточен (но в тоже время их не должно быть слишком мало);
- он не должен быть слишком простым или слишком сложным;
- все тестовые сценарии покрывают все основные функции в требованиях (ни одна функция разрабатываемой программы не должна остаться без внимания) [32].

В настоящее время наблюдается несколько методологий разработки тестовых сценариев, которые отличаются и теоретическим подходом, и практической реализацией. Проанализировав различные источники, можно сделать вывод, что наиболее часто употребляемая методология разработки тестовых сценариев — методология, при которой источниками сценариев выступают случаи использования. Случай использования – это некоторое множество сценариев, которые можно разделить на 3 группы: нормальный

случай, расширения и исключительные ситуации. Нормальный случай (или оптимистический сценарий) – это бизнес-сценарий, который выбирается чаще других. Расширение (или альтернативные пути) – это сценарии, отличающиеся от нормальных случаев в различных аспектах, но в то же время остаются полноценными путями исполнения. Исключительные ситуации – это те сценарии, которые приводят к возникновению ошибок. Каждый сценарий предусматривает действия, требуемые от тестируемой системы отклики, которые соответствуют основной части тестового сценария. Тестовый сценарий состоит из набора предусловий, входных данных и ожидаемого результата (отклика).

В разработку сценария для случая использования входит четыре действия:

1) идентификация всех вводных значений;
2) выделение классов эквивалентности для всех типов вводных значений;
3) составление списка комбинаций значений из разных классов эквивалентности;
4) построение тестовых случаев, в которых сочетаются одна перестановка значений с необходимыми внешними ограничениями [28].

Помимо этого, в тестовый сценарий входят предварительные условия – это условия, которые показывают готовность разрабатываемого ПО к проведению тестирования. Примерами предварительных условий являются наличие сборки на тестовом окружении, настройка, включение или выключение каких-либо функций на сервере и т.п.

Также существует несколько вариантов результатов запуска каждого сценария:

– пройден – ожидаемый результат совпадает с реальным;
– не пройден – ожидаемый результат не совпадает с реальным;
– пройден с незначительными ошибками – ожидаемый результат не совпадает с реальным, однако это не блокирует процесс (например, недочеты в удобстве пользования).

Таким образом, тестовый сценарий – это тестовая документация, в которой записаны стандартные и альтернативные сценарии работы с приложением, описание последовательности действий в системе и ожидаемого поведения. Тестовые сценарии отражают требования к программе согласно технической документации, в случае изменения которой, должны редактироваться и сценарии. Существуют некоторые классификации тестовых сценариев, одна из самых популярных – объединение сценариев по направлению, другими словами по позитивным случаям, негативным и альтернативным.

1.2.3 Выполнение тестов

Стоит отметить, что прежде чем приступить к выполнению тестирования, нужно убедиться, что предварительные условия тестового сценария выполнены.

Итак, на этапе тестирования выполняется итеративный запуск тестовых сценариев. Как правило, итерация состоит из шагов, представленных на рисунке 1.2.

Рисунок 1.2 – Этапы выполнения теста

Количество итераций тестирования ограниченно сроками сдачи проекта и используемыми стандартами качества. Тестирование может быть закончено после выполнения итерации без ошибок, при достижении требуемого объема выполненных тестов без обнаружения ошибок или по истечении выделенных временных ресурсов. В случае наличия не исправленных ошибок в момент завершения этапа формируется итоговый список [23].

В качестве основных этапов тестирования ПО можно выделить следующие:

- дымовое и санитарное тестирование – предварительная проверка разрабатываемого ПО, основные и базовые поверки. На этом этапе задача тестировщика убедиться, что тестовая среда настроена и работает, а полученная сборка содержит необходимый функционал или изменения;
- функциональное и нефункциональное тестирование – основной этап тестирования, который включает в себя тестирование всех функций системы в соответствии с документацией, а также тестирование стабильности, удобства, совместимости программы и т.п., а также иногда сюда включают тестирование документации;
- регрессионное тестирование – проверка, что новые изменения не сломали уже существующий функционал, или проверка на то, что функционал готов к выдаче;
- интеграционное и end-to-end тестирование – проверка того, как разрабатываемый модуль, система или продукт взаимодействует с другими модулями, системами и продуктами. На данном этапе проверяется вся цепочка действий пользователя при работе с системой. Например, пользователь делает заявку через сайт интернет магазина, далее заявка записывается в базу, далее обрабатывается и передается в систему для закупок и т.д. В этом случае тестировщик должен настроить необходимое окружение и проверить весь жизненный цикл заявки;
- демо-тестирование и приемочное тестирование – этап демонстрации ПО заказчику или пользователям [14].

Более подробно вышеперечисленные виды тестирования будут рассмотрены в следующей главе.

Помимо этого, во время проведения тестирования тестировщики в 99% случаев находят ошибки, которые заносятся в специальную систему отслеживания. Помимо создания и описания ошибки, тестировщик должен проинформировать разработчика о найденной ошибке, а также дождаться ее исправления. После того как разработчик исправил ее, тестировщик повторят проверку, и либо закрывает ошибку, если она исчезла, либо переоткрывает, если ошибка все еще не исправлена.

Таким образом, выполнение тестов – основной этап тестирования, который состоит из итеративных запусков тестовых наборов. Каждая итерация состоит из выполнения теста, обновления тестовых наборов, приемочных испытаний, запуска основного набора тестов и анализа результатов тестирования. В свою очередь этап выполнения тестов также делится на под этапы: дымовое и санитарное тестирование, функциональное и нефункциональное тестирование, регрессионное тестирование, интеграционное и end-to-end тестирование, а также демо-тестирование и приемочное тестирование.

1.2.4 Анализ результатов тестирования и выдача проекта

На основе предыдущих этапов создаётся документ, который описывает не только результаты проведенных тестов, но и ход выполнения каждого из них.

Отчёт по тестированию содержит следующую информацию: цель проведения тестирования, конфигурация тестового стенда и генератора нагрузки, требования к ПО, тестовые сценарии, данные о максимально возможном количестве одновременно работающих пользователей, сведения о количестве и типах ошибок, выводы о производительности системы в целом и о слабых местах, если они были обнаружены [2].

После этого наступает этап выдачи, на котором тестировщик также играет важную роль, так как помимо написания руководства по установке новой сборки, он также пишет руководства пользования, например, если для включения какого-либо функционала нужно что-то настроить. Настройками на реальном сервере занимаются специальные люди, задача тестировщика – подробно описать, как это сделать.

Таким образом, на этапе анализа результатов принимается решение о внедрении ПО, либо оно уже готово к использованию, либо необходимы исправления и доработки. На данном этапе создается документ, который отражает результаты и ход тестирования. А после этого происходит выдача проекта, для которой создаются специальные руководства пользования.

1.3 Определение понятия «методика тестирования»

В различных источниках дается множество определений такому понятию как «методика», однако во всех них есть что-то общее. Проанализировав данные определения, можно сделать вывод, что методика – это алгоритм определенных действий, который в итоге приводит к заранее определенному результату. Методика играет важную роль в эмпирическом исследовании, и в отличие от метода, в задачи методики не входит теоретическое обоснование полученного результата, т.к. она концентрируется на технической стороне эксперимента [33]. В данном пункте стоит отметить. Что многие ученые забывают о том, что метод – это чисто теоретическое описание чего-либо, в то время как методика придерживается практической обоснованности.

«Методическая корректность исследования гарантирует воспроизводимость результата исследования, возможность его контроля и проверки» [28]. В современном мире оборудование и техника усложнились, практическое осуществление всех требований часто невозможно или влечет за собой огромные расходы, поэтому все большее значение приобретает детальное описание методической стороны исследования.

В последнее время обострилось внимание к методике, как к обеспечивающей сфере научного познания из-за расширения инструментальной и расчетной базы естественных и социальных, что прежде всего проявляется в стандартизации методик. Практически все рутинные исследования проводятся по стандартным методикам, описанным в справочниках. В результате, ученые отдельных дисциплин начали обсуждать и критиковать методический инструментарий соответствующих дисциплин. Но в то же время прогресс в сфере методологической деятельности внес свой вклад в интеграцию разных областей, так как методики, разработанные в одной сфере, часто приобретают универсальный характер и могут использоваться в другой.

При составлении методики нельзя забывать о требованиях, которым она должна соответствовать:

- реалистичность;
- воспроизводимость;
- внятность;
- соответствие целям и задачам планируемого действия;
- результативность [28].

В области информационных технологий также не обошлось без своих методик, однако мы остановимся конкретно на области тестирования. Проанализировав большое количество теоретического материала по тестированию программного обеспечения, мы пришли к выводу, что не существует универсальных методик тестирования, в то время как существует большое количество методов тестирования (которые также иногда называются методиками, не являясь таковыми). В то же время конкретные шаги, которые применяются при тестировании, и которые могли бы принадлежать какой-то из методик, не принадлежат ни одной из них, и существуют отдельно друг от друга. Таким образом за неимением конкретного определения, что же такое «методика тестирования», мы вывели собственное определение, базируясь на выше полученной информации.

Методика тестирования – это приемы и конкретные действия, направленные на выявление того, насколько реализованный функционал разрабатываемого ПО соответствует заявленным требованиям.

Методика тестирования должна также соответствовать требованиям, т.е. она должна быть:

1) воспроизводимой – подходить для большего количества программных обеспечений и приложений;
2) понятной для большинства практикующих тестировщиков;
3) полной – большинство функций приложения могут быть протестированы шагами из данной методики, а также она должна покрывать большинство видов тестирования;
4) результативной – после тестирования по данной методике, тестировщик должен получить конкретные результаты.

Помимо этого, для составления корректной методики тестирования нужно рассмотреть популярные методы тестирования, что поможет выявить их достоинства, на основе которых и будут выявлены конкретные действия, подходящие под новую методику. Таким образом, можно сделать вывод, что в настоящее время не существует конкретного определения к термину «методика тестирования», что осложняет работу практикующих (в особенности начинающих) тестировщиков, так как информация о том, что нужно сделать, для того чтобы протестировать определенных программный продукт разнесена по разным источникам. В результате анализа различных источников было выведено собственное определение данному термину, т.е. методика тестирования – конкретные действия, направленные на выявление того насколько реализованный функционал разрабатываемого приложения обеспечения соответствует заявленным требованиям.

Выводы по первой главе

В результате выполнения первой главы можно сделать вывод, что тестирование – это процесс оценки системы или ее компонентов с намерением

определить, удовлетворяет ли она указанным требованиям или нет. Простыми словами, тестирование идентифицирует любые пробелы, ошибки или отсутствующие требования в противоположность фактическим требованиям.

Существуют различные этапы тестирования, первым из которых является планирование тестирования, т.е. согласование тематики проекта с заказчиком, выработка идей проекта, сбор и анализ требований, определение функциональных характеристик продукта. Следующим этапом является разработка тестовых сценариев, т.е. описание последовательности действий в системе и ожидаемого поведения будущей программы. Тестовые сценарии отражают технические требования к программе, в случае корректировки которых, должны быть дополнены и составленные сценарии. Следующий этап – проведение тестов, т.е. итеративный запуск тестовых сценариев. Помимо этого, этап выполнения тестов также делится на под этапы: дымовое и санитарное тестирование, функциональное и нефункциональное тестирование, регрессионное тестирование, интеграционное и end-to-end тестирование, а также демо-тестирование и приемочное тестирование. Последним этапом тестирования является анализ результатов, на котором принимается решение о внедрении ПО, на данном этапе создается документ, который результаты и ход тестирования.

Кроме того, существует множество определений термину «методика», однако нами не было выявлено ни конкретного определения для термина «методика тестирования» ни конкретной методики тестирования, которой можно следовать в процессе работы. Впрочем, проанализировав различные источники было представлено новое определение данному термину – определенные приемы и действия для достижения контроля качества программного обеспечения. Методика тестирования должна быть понятной практикующим тестировщикам, реалистичной, воспроизводимой для разных приложений, а также результативной, т.е. в результате следования по данной методике – тестировщик получит измеримые результаты своей работы.

Глава 2 Анализ существующих методов и типов тестирования программного обеспечения

2.1 Обзор существующих видов тестирования

Тестирование можно классифицировать по очень большому количеству признаков. В разных источниках приводится различные классификации видов тестирования, однако мы рассмотрим лишь некоторые из них.

В данной главе собраны и описаны большинство видов тестирования, что позволяет систематизировать и расширить знания и значительно ускорить процессы планирования тестирования и разработки тестовых сценариев, а также оптимизировать трудозатраты. После анализа приведенных ниже видов тестирования, можно будет сделать выводы, какие из них обязательно нужно будет покрыть будущей методикой тестирования, а какие из них могут остаться в стороне. Также стоит отметить, что мы не утверждаем, что те виды тестирования, которые не будут покрыты будущей методикой неважны, просто следует отметить, что невозможно в равной степени одной методикой покрыть все существующие виды, так как каждый из них направлен на разные аспекты тестирования ПО, а, как мы помним из предыдущей главы, методика должна быть простой и понятной, не трудозатратной и не перегруженной.

Одна из классификаций видов тестирования базируется на уровне детализации работ проекта, в нее входят: модульное, интеграционное, системное и приемочное тестирование.

Модульное тестирование – уровень, на котором осуществляется тестирование минимально возможного функционала для тестирования компонента, например, отдельного класса или функции. На этом уровне используются метод и приемы «белого ящика», которые будут рассмотрены в следующем пункте [14].

Модульное тестирование нацелено на независимую проверку работы компонент ПО. Тестирование модулей выполняется изолированно, без интеграции с другими модулями, в связи с чем для выполнения тестирования требуется реализовывать и/или подключать заглушки, эмуляторы и другие

вспомогательные инструменты, заменяющие полностью или частично реальные компоненты ПО.

Задачами модульного тестирования являются поиск дефектов, связанных с алгоритмическими ошибками, ошибками кодирования алгоритмов, выполнением условных и циклических операторов, использованием переменных и ресурсов [18].

В современных проектах все чаще модульное тестирование выполняется разработчиками, поэтому можно сделать вывод, что данный вид тестирования можно не покрывать будущей методикой, так как когда тестировщик будет приступать к своей работе отдельные модули будут уже протестированы командой разработчиков, а избыточность тестов в новой методике не приветствуется.

Интеграционное тестирование – уровень, на котором отдельные программные модули объединяются и тестируются в группе. Обычно интеграционное тестирование проводится после модульного тестирования [18].

С помощью интеграционных тестов можно проверить работу компонентов приложения на уровне, который включает инфраструктуру, поддерживающую приложение, такую как база данных, файловая система и сеть.

Интеграционные тесты служат для оценки компонентов приложения на более широком уровне, чем модульные. Данные тесты позволяют убедиться, что два или несколько компонентов разрабатываемой программы работают совместно для получения ожидаемого результата.

В отличие от модульных тестов, интеграционные тесты используют фактические компоненты (а не заглушки), которые приложение использует в рабочей среде [45].

Таким образом, можно сделать вывод, что данный вид тестирования должен быть покрыт новой методикой, так как, во-первых, разработчики редко тестируют интеграцию компонентов в приложения, а, во-вторых, часто бывают случаи, когда разные компоненты разрабатываются разными IT-компаниями, и

готовый продукт появляется только уже на тестовом окружении, а проверить интеграцию отдельных компонентов до полного системного тестирования очень важно, так как это снижает количество ошибок в будущем.

Системное тестирование – это тестирование программного обеспечения, выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям. Также стоит отметить, что системное тестирование не требует знаний о внутреннем устройстве системы [18].

Выделяют два подхода к системному тестированию:

– на базе функциональных и нефункциональных требований, для каждого из которых создаются тестовые сценарии;
– на базе случаев использования, на основе которых создаются пользовательские сценарии, а в последствие и тестовые.

Системное тестирование проводится следующим образом:

1) создание тестового плана, пользовательских и тестовых сценариев:
2) создание тестовых данных, используемых для системного тестирования:
3) проведение автоматизированного прогона тестовых сценариев, затем ручного.

По результатам тестирования составляется отчет по выявленным ошибкам и проводится регрессионное тестирование после их исправления. Цикл повторяется до исключения всех ошибок.

Таким образом, мы делаем вывод, что данный вид тестирования должен быть покрыт будущей методикой тестирования, так как системное тестирование часто подразумевает end-to-end процесс или тестирование бизнес-сценариев, а это является очень важной частью тестирования.

Тестирование пользовательского (графического) интерфейса, которое часто является частью системного тестирования – тестирование графических изображений и элементов (меню, кнопки, списки). При проведении данного тестирования тестировщики проверяют общий вид приложения.

Многие тестировщики выделяют следующие критерии тестирования графического интерфейса:

- вид элементов при уменьшении окна браузера;
- правильность написания и выравнивание текста;
- правильность перемещения фокуса с использованием клавиш Tab или Tab+Shift;
- возможность использования горячих клавиш;
- возможность выделения отдельных элементов/ текста;
- проверка наличия нужных нотификаций;
- единообразие дизайна (цвет, шрифт, размер);
- кнопки переключаются и нажимаются;
- проверка чекбоксов;
- выпадающие списки – выпадают и содержат список, возможность прокрутки длинного списка, выбранный элемент должен отображаться;
- проверка всплывающих окон, они не должны мешаться или раздражать пользователя, появляться только в случае необходимости, возможность закрытия [45].

Приемочное тестирование – это тестирование готового продукта конечными пользователями на реальном окружении, в котором будет функционировать тестируемое приложение. Приемочные тесты разрабатываются пользователями, обычно, в виде сценариев. Для того, чтобы найти больше ошибок рекомендуется планировать не только системное и приемочное тестирование, но и модульное, и интеграционное [11].

Проанализировав данный вид тестирования, можно сделать вывод, что он может быть исключен из новой методики, так как он похож на системное тестирование, только проводится конечными пользователями или отдельно нанятыми тестировщиками со стороны заказчика.

Следующая классификация разделяет тестирование по запуску кода на исполнение. Одним из видов тестирования согласно данной классификации является статическое тестирование – это вид тестирования, где

разрабатываемое ПО проверяется без запуска кода. Кроме этого, оно находит и устраняет ошибки в разного рода сопроводительных документах, например, специфики требований к ПО [32].

Статическое тестирование бывает двух типов: обзоры и статический анализ.

Обзоры – тестирование, направленное на обнаружение дефектов в документации, которое также классифицируются на:

- инспектирование программного обеспечения – подразумевает проверку документации высшими органами, например, изучение требований к ПО;
- неформальные обзоры – созданный документ демонстрируется публике, где каждый высказывает свое мнение на счет него, что помогает обнаружить недочеты на ранних стадиях создания продукта;
- оценка экспертов – проверка документов командой специалистов с целью выявить и устранить ошибки;
- сквозные просмотры – проверка документа опытным специалистом на наличие дефектов.

Статическое тестирование может проводиться на всех этапах жизненного цикла ПО, т.е. на этапе планирования, сборе и анализе требований, проектировании, реализации, тестировании и сопровождении [17].

Статический анализ – анализ написанного кода на наличие недоработок в структуре, способных привести к багам.

В состав статического анализа входит оценка качества написанного кода. Для анализа кодовой комбинации и сравнения его со стандартами соответствия, применяют различные инструменты [25].

С помощью статического анализа можно обнаружить, например, мертвый код, неиспользуемые переменные, неправильный синтаксис, неопределяемые значения или нескончаемые циклы.

Преимуществами метода статического тестирования являются:

- возможность нахождения ошибок на первичных этапах разработки ПО, что способствует снижению стоимости исправления обнаруженных дефектов;

– возможность усовершенствования функциональной стороны процесса благодаря оставленным в процессе тестирования отзывам;
– высокий уровень информативности о проблемах качества ПО;
– улучшение обмена важной информацией между сотрудниками [11].

Проанализировав данный метод, можно сделать вывод, что по крайней мере его часть про тестирование документации должна быть покрыта новой методикой тестирования.

Следующий вид тестирования, согласно данной классификации, динамическое тестирование, в ходе которого выполняется проверка тестируемой программы (кода).

Динамическое тестирование направлено на проверку функционала программы во время выполнения кода. Данный вид тестирования подразумевает фактическую эксплуатацию программы и определение того, как работает ее функционал, в соответствии с ожиданиями или нет [21].

Динамический вид тестирования состоит из непосредственного тестирования программного обеспечения в реальное время, способом предоставления входной информации и исследования полученного результата поведения приложения.

Данный метод тестирования помогает проверить разные критические моменты программного обеспечения. Если закрыть глаза на их существование и никак не отреагировать на них, это может определенным образом сказаться на производительности, функциональной стороне и надежности приложения [3].

Преимуществами динамического тестирования можно считать:
– в процессе тестирования проводится тщательное изучение всего функционала программы, в результате чего получаем высокое качество проверки;
– это хорошо структурированный процесс, осуществляющий проверку программы со стороны пользователя, что, в свою очередь, значительно повышает качество программного обеспечения;

– исправление сложных дефектов, которые могли остаться незамеченными на этапе проверки кода;
– возможность автоматизации [14].

Можно выделить следующие виды динамического тестирования: функциональное тестирование, тестирование безопасности, тестирование взаимодействия, тестирование производительности, тестирование установки, тестирование удобства пользования, тестирование на отказ и восстановление, конфигурационное тестирование, дымовое тестирование, регрессионное тестирование, повторное тестирование, тестирование сборки, санитарное тестирование [38], что дает нам понять, что динамическое тестирование является «клоном» системного тестирования, просто в другой классификации и с другим названием, поэтому оно также должно быть покрыто будущей методикой.

В ходе функционального тестирования проверяется соответствие работы приложения ожиданиям пользователя. Проверка выполняется на основе предъявляемых к приложению функциональных требований и ограничений [9].

Данный вид тестирования ПО помогает:
– проверить правильность работы приложения при различных сценариях использования;
– проверить соответствие продукта требованиям;
– проанализировать уровень качества ПО и приоритизировать найденные дефекты по степени критичности.

Для получения полной картины о работе программы используется три вида тестирования:
– ручное тестирование: подготовка и проведение тестов в ручном режиме;
– автоматизированное: разработка, запуск и поддержка автотестов;
– исследовательское: комплексное изучение продукта и выявление дефектов за пределами требований.

Ключевым преимуществом является то, что данное тестирование на 100% покрывает тестовые сценарии и заложенные требования. Такое тестирование

следует проводить даже при отсутствии видимых проблем в работе продукта, чтобы гарантировать его дальнейшую правильную работу.

Тестирование взаимодействия относится к категории функционального тестирования и проверяет способность приложения взаимодействовать с одним и более компонентами или системами. Данный вид тестирования включает в себя тестирование совместимости и интеграционное тестирование, которое было описано выше [21].

Процесс тестирования производительности – это особые технические манипуляции, в ходе реализации которых проверяется качество продукта, а также его способности стабильно функционировать под определенной нагрузкой. Данная процедура представляет собой базовый нефункциональный метод тестирования, при котором оценивается текущая возможность ПО нормально работать с точки зрения стабильности и откликов при обыкновенной нагрузке. [44].

Процесс тестирования установки, в первую очередь, ориентирован на тестирование корректности инсталляции ПО и настройки, а также последующее обнаружение обновлений или удаление программного обеспечения.

Тестирование удобства пользования направлено на установление степени удобства использования, обучаемости, понятности и привлекательности ПО для конечных пользователей. Данный вид тестирования является одним из важнейших, так как предполагает проверку того, насколько просто и удобно конечным пользователям взаимодействовать с приложением. Также не стоит забывать про тестирование обучаемости, т.е. ощущение, испытываемое пользователем во время использования цифрового продукта, его способность к освоению, запоминанию и пониманию работы ПО [38].

Дымовое тестирование – это особый вид проверки программного обеспечения, направленный на то, что все самые важные параметры тестируемого продукта функционируют корректно и согласно заявленным ожиданиям, после того как в работу тестировщикам поступила новая сборка с новым функционалом.

Санитарное тестирование – очень специфическая проверка работоспособности отдельных функциональных элементов, систем, веб-архитектур и расчетов. Она выполняется с единственной целью – дать 100% понятие того, что создаваемая система работает именно так, как того и требовалось. Подобный тип тестирования нередко выполняется до старта полного цикла проведения регрессионных проверок, но только после окончания дымового тестирования.

Традиционно, санитарные проверки выполняются тогда, когда исправлен незначительный дефект внутри системы или ранее выполнялись работы по частичном редактировании логики работы ПО.

Как только изменения были внесены в код, новая сборка программного обеспечения передается на проверку в отдел тестирования. После установки сборки QA-специалисты могут выполнять эффективную проверку отредактированной функциональности вместо полной регрессии программного обеспечения, которое занимает существенно больше времени [44].

Регрессионное тестирование также должно быть неотъемлемой частью любого процесса разработки ПО. Этот вид тестирования включает все виды тестирования, которые выполняются на уже проверенных функциональных и нефункциональных областях программного обеспечения после исправления ошибок или каких-либо других улучшений и дополнений в коде.

Регрессионное тестирование осуществляется после исправления задокументированных дефектов, добавления в приложение новых функций и возможностей, а также после изменений в требованиях к программному обеспечению и приведения ПО в соответствие с новыми требованиями.

Регрессионные тесты надо постоянно дополнять и исправлять, чтобы обеспечить достаточное покрытие недавно добавленных частей программного обеспечения. Стоит отметить, что данный вид тестирования в основном автоматизируется, так как оно содержит много повторяющихся время от времени тестов [35].

Стоит отметить, что, так как вышеперечисленные виды принадлежат к динамическому тестированию, из этого следует вывод, что практически все из них должны быть покрыты новой методикой.

Таким образом, существует большое количество классификаций и видов тестирования, например, статическое и динамическое тестирование, модульное и интеграционное тестирование, системное тестирование и т.п. Проанализировав данные виды, мы сделали выводы, какие из них должны быть покрыты новой методикой тестирования, а какие можно опустить.

2.2 Обзор существующих методов тестирования

Как уже было сказано в предыдущей главе, тестирование является частью верификации программной системы, т.е. достижения гарантии того, что верифицируемый объект соответствует всем требованиям [27].

Существует множество классификаций тестирования, некоторые из которых были описаны в предыдущем пункте. Одной из самых распространенных классификаций является классификация по доступу к коду и архитектуре, которую мы рассмотрим ниже.

Согласно стандарту IEEE 1008-87, существует два основных метода тестирования, основанные на том, имеет ли разработчик тестов или тестировщик доступ к исходному коду тестируемого ПО, или же тестирование выполняется через пользовательский интерфейс: метод белого и черного ящика соответственно. Данные методы формируют часть режима тестирования программного обеспечения. Эти тесты обычно считаются самодостаточными в поиске ошибок во всей системе [26].

2.2.1 Метод черного ящика

Тестирование методом черного ящика осуществляется без каких-либо знаний внутренней работы системы. Это тип тестирования, в котором функциональные возможности программного обеспечения тестируются без каких-либо ссылок на внутренний дизайн, код или алгоритм, используемый в

программе. В наше время многие компании для получения точных результатов передают испытание своих работ третьим лицам. Это происходит потому, что разработчик системы очень хорошо понимает внутреннюю логику и кодирование системы, что делает его непригодным для тестирования его приложения данным методом [6].

Каждый метод имеет свои идеи, концепции и техники тестирования, в методе черного ящика выделяют четыре основных техники: эквивалентное разбиение, анализ граничных значений, функциональные диаграмма и предположение обо ошибке.

Техника эквивалентного разбиения включает в себя разделение входных значений на допустимые и недопустимые и выбор репрезентативных значений из каждого раздела в качестве тестовых данных. С помощью данной техники сокращается количество тестовых сценариев, а также сохраняется приемлемое тестовое покрытие.

Допустим, нужно протестировать поле ввода «возраст». Согласно требованиям, диапазон чисел для данного поля от 0 до 100. Согласно данной технике, мы должны провести как минимум 5 тестов: [любое значение внутри интервала], [от -∞ до 0], [от 101 до + ∞], [специальные символы], [буквы]. Конечно, данными тестами можно не ограничиваться, но бесспорный плюс данной техники состоит в том, что она отсеивает огромное количество значений ввода, что сокращает время на тестирование [20].

С помощью техники анализа граничных значений можно определить границы входных значений и выбрать в качестве тестовых данных значений те, которые находятся на границах, внутри и вне границ. Многие системы имеют тенденцию вести себя некорректно при граничных значениях, поэтому оценка значений границ приложения очень важна. Количество тестов для проверки граничных значений будет равно количеству границ, умноженному на 3, так как рекомендуется проверять значения вплотную к границе, следовательно, если, например, тестировщик хочет проверить границу значения 100, его тестовые сценарии будут: 101, 100, 99 [30].

Недостатком предыдущих двух техник является то, что они не исследуют комбинации входных условий, так как даже при построенном эквивалентном разбиении входных условий число комбинаций обычно очень велико. Если нет систематического способа выбора подмножества входных условий, то, как правило, выбирается произвольное подмножество, приводящее к неэффективному тесту.

Техника функциональных диаграмм или диаграмм причинно-следственных связей помогает систематически выбирать высоко результативные тесты. Она позволяет увидеть неполноту и неоднозначность исходных спецификаций. Для понимания данной техники вовсе не обязательно знание электроники, но желательно понимание логики логических операторов.

Построение тестов этим методом осуществляется в несколько этапов: сначала входные данные разбиваются на классы эквивалентности, затем определяются причины и следствия, т.е. входные значение (или классы эквивалентности) и выходные значения (или преобразование программы – действие, которое входное условие оказывает на состояние программы), помимо этого каждой причине и следствию присваивается номер, а затем причины и следствия преобразуются в булевский граф, что и является функциональной диаграммой. Диаграмма добавляется примечаниями, в которых описываются ограничения и комбинации причин и (или) следствий, которые являются невозможными по тем или иным причинам, а затем преобразуется в таблицу решений с ограниченными входами, каждый столбец которой соответствует тесту. Столбцы таблицы решений преобразуются в тесты.

Базовые символы для записи функциональных диаграмм показаны на рисунке 2.1.

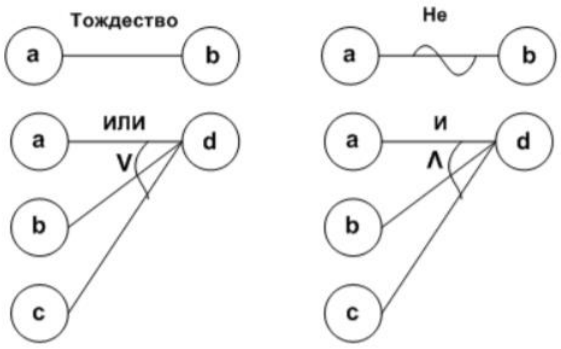


Рисунок 2.1 – Базовые логические отношения функциональных диаграмм

Каждый узел диаграммы может находиться в двух состояниях: 0 («отсутствует») или 1 («присутствует») [3].

Последняя техника тестирования в методе черного ящика - предположение об ошибке, заключается в том, что тестировщик с большим опытом работы выискивает ошибки без всяких методик, но при этом он подсознательно использует технику предположения об ошибке. Данный метод в значительной степени основан на интуиции. Основная идея техники заключается в том, чтобы составить список, который перечисляет возможные ошибки и ситуации, в которых эти ошибки могли проявиться. Потом на основе списка составляются тесты [26].

С помощью метода «черного ящика» можно провести множество различных тестов, таких как функциональное тестирование, стресс-тестирование, тестирование удобства, приемочное или регрессионное тестирование.

Однако, по мнению таких авторов как Петров А.С., Плаксин М.А. и Сергеев Д.И., данный метод является наиболее полезным при интеграционном тестировании, так как в данном случае программные и аппаратные компоненты объединяются и тестируются для оценки взаимодействия между ними. При использовании метода «черного ящика» тестировщик проверяет, корректно ли работают все компоненты в целом, когда они интегрированы в большую систему, т.к. нормальная работа каждой составляющей по отдельности – это

еще не гарантия того, что они будут работать вместе в рамках всего проекта. Например, данные могут не отправиться через интерфейс, или интерфейс не отработает согласно документации [34].

У метода «черного ящика», как и у любого другого, есть свои достоинства и недостатки. К достоинствам можно отнести:

1) доступность: разрабатывать тест-кейсы можно сразу же после завершения работы со спецификацией; тестирование данным методом применимо всегда и не зависит от того, доступен ли исходный код.
2) воспроизводимость: метод не требует привязки к конкретному приложению или программе, поэтому составленные тест-кейсы могут быть адаптированы и к другим тестируемым системам.
3) простота: проверка проходит с позиции конечного потребителя; тестировщику не обязательно обладать узкопрофильной специальностью или знанием кода.

Проанализировав данные положительные особенности можно сказать, обязательные характеристики для новой методики, описанные в первой главе, совпадают с некоторыми из вышеперечисленных достоинств данного метода, например, доступность и воспроизводимость.

Однако несмотря на свои достоинства, данный метод не может быть универсальным, так как существует ряд недостатков, с которыми сталкиваются практикующие тестировщики. Не всегда данный метод тестирования является достаточным, ведь всегда есть риск пропустить серьезные ошибки. Есть возможность пропуска границ и переходов, которые неочевидны из спецификации, но есть в реализации кода. С использованием данного метода можно протестировать только небольшое количество возможных вводных (входящих) значений; многие варианты остаются без проверки. Также тесты могут быть избыточными, если разработчик уже проверил ту или иную функциональность. При отсутствии четкой и полной спецификации проектировать тесты и тест-сценарии оказывается затруднительно [37].

Таким образом, метод черного ящика – тестирование без каких-либо знаний внутренней работы системы. Тестировщик стимулирует программное обеспечение для пользовательской среды, предоставляя различные входные значения и тестируя сгенерированные выходы. Анализируя данный метод для будущей методики, можно сказать, что он подходит под такие пункты как простота и воспроизводимость.

2.2.2 Метод белого ящика

Тестирование «белого ящика» – особый метод проверки ПО, который подразумевает, что внутренняя структура и технические особенности ПО досконально известны проверяющему. Проверка белого ящика состоит из нескольких взаимодополняющих типов тестирования, используемых для оценки удобства применения веб-продукта, части кода или особого программного функционала.

Основным методом тестирования «белого ящика» является анализ покрытия кода. В настоящее время существуют некоторые инструменты, которые позволяют проанализировать, в какие строки были вхождения во время тестирования, вследствие чего можно значительно повысить покрытие, добавить новые тесты для конкретных случаев, и кроме этого избавиться от дублирующих тестов. Проведение такого анализа и последующая оптимизация довольно легко воплощается в рамках метода белого ящика при модульном, интеграционном и системном тестировании, в то же время в методе черного ящика данная задача станет достаточно дорогостоящей, так как она затратна по времени и ресурсам, так как нужно установить, настроить и проанализировать результаты работы со стороны тестировщиков и разработчиков.

Помимо покрытия кода при тестировании методом «белого» ящика используется покрытие потоков управления. Данная техника основана на определении путей выполнения кода и создании выполняемых тестовых сценариев для покрытия этих путей [27].

Фундаментом для тестирования потоков управления является построение графов потоков управления, основными блоками которых являются:

– блок процесса – одна точка входа и одна точка выхода;
– точка альтернативы – одна точка входа, две и более точки выхода;
– точка соединения – две и более точек входа, одна точка выхода.

Для тестирования потоков управления определены разные уровни тестового покрытия, на которые можно посмотреть в таблице 2.1.

Основываясь на вышеприведенных данных, тестировщик сможет спланировать необходимый уровень тестового покрытия, а также оценить уже имеющийся [8].

Согласно Исаеву Г. в его работе «Проектирование информационных систем», существует три аспекта, которые проверяются стратегией белого ящика, а именно:

1) разработано ли программное обеспечение в соответствии с оригинальным дизайном;
2) внедрены ли надежные меры безопасности в программное обеспечение;
3) выяснены ли уязвимости данного программного обеспечения [26].

К основным достоинствам данного метода тестирования можно отнести:

1) оптимизацию программного кода путем поиска скрытых багов;
2) возможность создания автоматизированных тест-кейсов;
3) использование наиболее подходящего вида входных данных, применяемых для качественного процесса тестирования.

Лучше всего данный метод подходит для модульного тестирования, так как было сказано в одном из предыдущих пунктов данный вид тестирования направлен на поиск ошибок связанными с алгоритмами, условий, циклов и т.п.

Вспоминая обязательные характеристики будущей методики тестирования, можно сделать вывод, что метод белого ящика будет полезен для такой характеристики как «результативность», так как с помощью данного метода тестирования можно покрыть большое количество функционала.

Таблица 2.1 – Уровни тестового покрытия

Уровень	Название	Краткое описание
Уровень 0	-	“Тестируй все что можешь, остальное протестируют пользователи” (*“Test whatever you test, users will test the rest”)*
Уровень 1	Покрытие операторов	Каждый оператор выполнен не менее одного раза.
Уровень 2	Покрытие альтернатив/ ветвей	Каждый узел с ветвлением (альтернатива) выполнен не менее одного раза.
Уровень 3	Покрытие условий	Каждое условие, имеющее TRUE и FALSE на выходе, выполнено не менее одного раза.
Уровень 4	Покрытие условий альтернатив	Тестовые сценарии создаются для каждого условия и альтернативы.
Уровень 5	Покрытие множественных условий	Достигается покрытие альтернатив, условий и условий альтернатив (Уровни 2, 3 и 4)
Уровень 6	Покрытие бесконечного числа путей	Если, в случае зацикливания, количество путей становится бесконечным, допускается существенное их сокращение, ограничивая количество циклов выполнения, для уменьшения числа тестовых сценариев.
Уровень 7	Покрытие путей	Все пути должны быть проверены

Главным недостатком «белого ящика» является трудоемкость. Сложность связана с тестируемым приложением. Небольшое приложение, которое выполняет одну простую операцию, может быть протестировано в течение нескольких минут, в то время как более крупным программным приложениям требуются дни, недели и даже больше для полного тестирования. Тестирование белого ящика требует профессиональных ресурсов с подробным пониманием программирования и реализации. Это приводит к тому, что тестирование является дорогостоящим [34].

Таким образом, метод белого ящика – тестирование со знанием внутреннего функционирования и логики работы кода. Для выполнения данных тестов, тестировщик должен иметь знания кода, чтобы узнать точную место в коде, имеющее ошибки. Сравнивая данный метод с характеристиками будущей методики тестирования, можно сказать, что он подходит под такой пункт, как результативность, но не подходит из-за своей «трудозатратности».

2.2.3 Метод серого ящика

В предыдущих пунктах были приведены два основных метода тестирования, принятые стандартом IEEE 1008-87, однако многие тестировщики также выделяют еще один, третий, метод тестирования – метод «серого ящика», который часто представляют комбинацией первых двух методов.

Тестирование «серого ящика» – специальный метод тестирования ПО, когда тестировщик обладает неполным знанием внутреннего устройства системы. Этот метод тестирования также иногда называют методом полупрозрачного ящика: что-то видно, а что-то – нет.

Чтобы выполнить подобный вид тестов, не нужно иметь доступ к исходному коду ПО. Предполагается, например, доступ к внутренним механизмам и алгоритмам работы ПО для написания максимально комплексных тестовых сценариев, но само тестирование проводится с помощью метода черного ящика, то есть, с позиции пользователя. Или

тестировщик может полностью придерживаться метода черного ящика, но для того чтобы убедиться в правильной работе отдельных алгоритмов и механизмов он может смотреть информацию в логах или анализировать записи программы в базе данных.

Все тесты создаются на базе простого знания алгоритмов, архитектуры и иных высокоуровневых характеристик поведения разрабатываемого продукта [26].

Джем Канер, профессор в области программной инженерии, определяет тестирование серого ящика как «включающее входы и выходы, но дизайн теста основан на информации о коде или работе программы, которая обычно не видна тестировщику» [44].

Выделяют следующие методы тестирования серого ящика:

1) матричное тестирование: содержит отчет о состоянии проекта;
2) регрессионное тестирование: подразумевает повторный запуск тестовых случаев при внесении новых изменений;
3) тестирование по образцу: проверка приложения на предмет его дизайна, архитектуры и шаблонов;
4) проверка ортогонального массива: используется как подмножество всех возможных комбинаций.

Метод «серого ящика» подходит для тестирования web-приложений, так как они имеют распределенную сеть, а из-за отсутствия исходного кода невозможно использовать тестирование методом белого ящика. Тестирование черного ящика также не подойдет из-за простого контракта между заказчиком и разработчиком, поэтому более эффективно использовать тестирование серого ящика, так как значительная информация доступна на языке WSDL [5].

Помимо этого, тестирование методом «серого ящика» подходит для разных видов функционального тестирования, но лучше всего для интеграционного, так как оно проводится в основном для проверки взаимодействия пользователя с внешними системами. Данный метод также

помогает подтвердить, что программное обеспечение соответствует требованиям, определенным для программного обеспечения [30].

Также стоит отметить, что некоторые авторы определяют метод «серого ящика» как противопоставление методам белого и чёрного ящика, особо подчёркивая, что при работе по методу серого ящика внутренняя структура тестируемого объекта известна частично и выясняется по мере исследования. Этот подход, бесспорно, имеет право на существование, но в данной работе мы представляем данный метод комбинацией (а не противопоставлением) белого и чёрного ящиков.

В то же время по этой же причине некоторые авторы не признают существования данного метода, так как они считают, что комбинация существующих методов не может образовать новый, это просто комбинация и ничего более.

Основными преимуществами метода можно считать:

1) сочетание достоинств и черного, и белого ящиков, т.е. тестировщик смотрит на тестируемый объект с позиции метода черного ящика, но при этом проводит анализ на основе тех данных, что он знает о системе., то есть с помощью метода белого ящика;
2) возможность создания и применения более сложных тестовых сценариев, чем по методу черного ящика;
3) возможность совместной работы с разработчиком, что позволяет на начальном этапе убрать избыточные тестовые сценарии. Это сокращает время функционального и нефункционального тестирования;
4) предоставление разработчику достаточно времени для исправления дефектов.

Проанализировав данные достоинства, можно сказать, что данный метод более всего применим к составлению будущей методики тестирования, во-первых, так как он сочетает достоинства двух предыдущих методов, а, во-вторых, он менее трудоемкий.

Однако у данного метода есть и свои недостатки, например, ограничение во времени, так как тестирование всех потоков ввода и вывода информации занимает очень много времени, или избыточность тестов, когда, например, не только тестировщик, но и программист проверяет свой код с помощью модульных тестов, а также при тестировании данным методом у тестировщика ограничены возможности анализа покрытия кода из-за отсутствия доступа к нему или отсутствия нужных знаний [38].

Таким образом, многие (но не все) специалисты выделяют еще один метод тестирования – метод серого ящика, который является комбинацией первых двух методов. Для выполнения тестирования «серого ящика» нет необходимости в доступе к исходному коду, т.к. тесты пишутся на основе знания алгоритма, архитектуры, внутренних состояний или других высокоуровневых описаний поведения разрабатываемой программы. У данного метода есть свои достоинства и недостатки, но можно сделать вывод, что он более всего применим для будущей методики тестирования, благодаря тому, что он сочетает в себе достоинства черного и белого ящиков, а также, потому что он менее трудоемкий, чем последний.

Выводы по второй главе

В данной главе рассмотрены основные виды тестирования, а также было принято решение, должны ли они быть покрыты новой методикой тестирования. Таким образом, будущая методика должна покрывать интеграционное (тестирование на взаимодействие разных компонент), системное (тестирование полной программы), статическое (а именно тестирование документации), и динамическое, которое по своей сути очень похоже на системное и включает в себя множество подвидов, таких как функциональное тестирование, тестирование безопасности, тестирование удобства пользования и т.п.

Помимо этого, были рассмотрены три метода тестирования ПО. Это методы черного ящика, белого ящика и серого ящика.

Тестирование методом черного ящика осуществляется без каких-либо знаний внутренней работы системы. Тестировщик будет стимулировать программное обеспечение для пользовательской среды, предоставляя различные входы и тестируя сгенерированные выходы. Для новой методики он полезен своей доступностью и производительностью.

Тестирование методом белого ящика учитывает внутреннее функционирование и логику работы кода. Для выполнения этого теста, тестировщик должен иметь знания кода, чтобы узнать точную часть кода, имеющую ошибки. Для новой методики он полезен своей результативностью, однако не стоит забывать о его трудозатратности.

Третьим рассмотренным методом тестирования является метод серого ящика, и он по-нашему мнению, является самым оптимальным для будущей методики, так как он включает в себя достоинства черного и белого ящиков, т.е. согласно данному методу тестировщик может обладать лишь общими знаниями разрабатываемого продукта, необходимыми для выполнения теста. Эта проверка осуществляется посредством документации и схемы информационных потоков.

Глава 3 Методика тестирования программного обеспечения

3.1 Описание предлагаемой методики тестирования

В предыдущей главе было выявлено, что наиболее оптимальным методом тестирования является метод серого ящика, так как он включает в себя достоинства черного и белого ящиков, а именно:

1) тестировщик смотрит на тестируемый объект с позиции метода черного ящика, но при этом он обладает некоторыми знаниями о «внутренностях» системы, например, имеет доступ к логированию;
2) проводит анализ на основе тех данных, что он знает о системе., то есть с помощью метода белого ящика;
3) возможность создания и применения более сложных тестовых сценариев, чем по методу черного ящика;
4) возможность совместной работы с разработчиком, что позволяет убрать избыточные тестовые сценарии.

Помимо этого, нами было дано определение такому понятию, как «методика тестирования» – это алгоритм действий для выявления того, насколько реализованный функционал разрабатываемого ПО соответствует заявленным требованиям. Также мы определили характеристики, которыми должна обладать новая методика тестирования:

1) воспроизводимость;
2) понятность для большинства практикующих тестировщиков;
3) полнота;
4) результативность.

Таким образом, новая методика должна состоять из следующих шагов:

1) сбор требований о тестируемой системе;
2) анализ требований;
3) разработка тестовых сценариев;
4) интеграционное тестирование отдельных компонентов и параллельное фиксирование и отслеживание найденных ошибок;

5) тестирование пользовательского интерфейса и параллельное фиксирование и отслеживание найденных ошибок;
6) системное тестирование и параллельное фиксирование и отслеживание найденных ошибок;
7) регрессионное тестирование;
8) выдача.

Данная методика также может быть представлена в виде алгоритма, который покрывает все вышеперечисленные шаги (приложение 1).

Данная методика структурирует всю теорию о тестировании ПО в понятную инструкцию, а также покрывает многие виды тестирования.

Мы считаем, что данная методика отлично подходит под выделенные нами критерии, она воспроизводима для большинства IT-продуктов, понятна многим тестировщикам, в том числе и начинающим, она включает в себя многие виды тестирования, а также в результате следования данной методике тестировщик получит измеримые результаты своей работы.

Далее мы рассмотрим каждый из этих шагов на примере конкретной программы, а также будут сделаны выводы об эффективности описанной методики.

В данной работе за пример был взят один из проектов компании ООО «НетКрэкер». Данная телеком компания является второй по величине компанией мобильной и фиксированной интернет связи в своей стране. По состоянию на август 2020 года у нее 5,7 миллиона клиентов.

В качестве примера будет взят один из процессов в данной компании – заказ на предоставление мобильной связи и покупку телефона. В соображениях безопасности мы не можем приложить исходный код и реальные сообщения, однако будут приложены скриншоты и *примерные* сообщения с изменёнными тестовыми данными. Также стоит отметить, что скриншоты официального сайта будут предложены на английском языке, так как на данный момент не существует русифицированной версии сайта.

Таким образом, в данном пункте была представлена новая методика тестирования, состоящая из сбора и анализа требований, разработки тестовых сценариев, интеграционного и системного тестирования (а также фиксирования и отслеживание ошибок), тестирование пользовательского интерфейса, регрессионное тестирование и выдачи проекта заказчику.

3.1.1 Сбор требований о тестируемой системе

Согласно открытым источникам, компания «НетКрэкер» предоставляет BSS услуги для многих проектов, в том числе и для того, который мы приводим в пример. BSS (Business Support System, система поддержки бизнеса) – «прикладное программное обеспечение поддержки деловых процессов предприятия, замкнутое на взаимодействие с абонентами, прежде всего биллинг (выставление счетов, обработка платежей и т.п.), CRM (управление информацией об абонентах, поддержка взаимодействия с ними и т.п.), ERP-системы, применяемые для обеспечения процессов финансового менеджмента, бухгалтерского учёта, управления персоналом, проектами и основными фондами также относят к подкатегории BSS» [15].

Так как в данном случае мы будем рассматривать только начало процесса предоставления мобильной связи наш процесс будет построен следующим образом:

1) заказчик заходит на официальный сайт телеком-компании;
2) заказчик выбирает желаемый продукт (в данном случае новый телефон и тариф);
3) заказчик заполняет все необходимые поля;
4) система обрабатывает его запрос;
5) система отправляет запрос в смежные системы для дальнейшей обработки данных;
6) система возвращает номер заказа;
7) Происходит проверка платежеспособности пользователя;
8) Подтверждение платежеспособности пользователя;

9) Пользователь получает электронный чек;
10) Пользователь производит оплату;
11) Пользователь забирает (или получает на дом) свой заказ.

Согласно информации в первой главе данной работы требования бывают трех видов: бизнес-требований, требования пользователей и требования к продукту.

Применимо к данному случаю, можно сказать, что:

- *бизнес-требование* данного процесса – возможность сделать заказ и возможность его дальнейшей обработки;
- *требование пользователя* – сделать заказ;
- *требования к продукту*:
  - удобство пользования;
  - надежность и безопасность (частные данные заказчика не должны попасть в открытый доступ);
  - технические требования (передача сообщений между системами на языке JSON, определение параметров, которые должен будет заполнить покупатель, их точное название, место в шаблоне и т.п.).

Таким образом, нами был собраны бизнес-требования, требования пользователя, а также требования к продукту на процесс покупки мобильного телефона и тарифа: заказчик должен получить возможность зайти на сайт и сделать заказ для его дальнейшей обработки.

3.2 Тестирование и анализ требований

Как было сказано в первой главе, в ходе анализа требований должны быть достигнуты следующие цели:

- адекватность требований: в данном случае требования являются адекватными, заказчик хочет получить возможность оставления заявки на покупки телефона и предоставления мобильной связи. Бизнес логика, получившаяся в результате сбора требований, устроила заказчика и соответствует его ожиданиям;

- полнота требований: в данном случае можно сделать вывод, что требования не полные, поэтому на данном этапе тестировщики и бизнес-аналитики должны уточнить следующие моменты:
  - Требование к браузеру;
  - Какие конкретные поля должны быть на странице заказа;
  - Какие поля должны быть обязательными для заполнения;
  - Каким образом должен отображаться номер заказа;
- совместимость требований: в данном случае функции друг другу не противоречат ни по логическим, ни по концептуальным компонентам;
- выполнимость и разумность требований: в данном случае группа тестирования пришла к выводу, что требования являются выполнимыми в поставленный срок;
- подверженность тестированию: соответствующая техническая документация составлена и соответствует всем требованиям заказчика. Недочёты и неясные моменты выяснены и внесены в соответствующую документацию.

Таким образом, нами были протестированы требования заказчика по всем основным пунктам, и можно сделать вывод, что требования являются адекватными, но изначально требовали доработки, так как существовало много неясных моментов, которые в последствии могли стать камнем преткновения, однако в процессе обсуждения данных неточностей, все вопросы были урегулированы и занесены в соответствующую документацию, также требования являются выполнимыми, разумными, а ожидаемый результат подвержен тестированию.

3.2.1 Разработка тестовых сценариев

Согласно проведенному анализу в первой главе, самой распространенной методологией разработки тестовых сценариев является методология на основе случаев использования.

В данном случае мы будем составлять сценарии для заполнения «анкеты», которую должен заполнить покупатель, чтобы оставить заказ. Так как данная анкета содержит слишком большое количество полей, мы не будем составлять тестовые сценарии для каждого из них, мы возьмем несколько полей с различными вводными значениями и напишем тестовые сценарии для них.

1) Радиокнопка – для выбора типа заказа (“Privately” или “For Business”):

В данном случае будет целесообразно прописать как минимум три тестовых сценария:

Нормальный случай: пользователь выбирает значение “Privately” → ожидаемый результат: система принимает данное значение и обрабатывает его.

Альтернативный случай: пользователь выбирает значение “For Business” → ожидаемый результат: система принимает данное значение и обрабатывает его.

Исключительный случай: пользователь ничего не выбирает → ожидаемый результат: система не дает оставить поле пустым, один из вариантов обязательно должен быть выбран.

Помимо группы тестирования, требования анализируют и разработчики, поэтому после их анализа тестировщик может проверить шаблоны сообщений, которые они собираются отправлять в смежные системы для дальнейшей обработки. Еще раз стоит отметить, что из-за контракта о неразглашении коммерческой тайны мы не можем прикрепить точные сообщения, которые получаются в результате заполнения формы пользователем, однако мы можем привести немного измененное сообщение, которое будет подходить под данную ситуацию.

Давайте рассмотрим первую часть сообщения, которая относится к вышеуказанному параметру (листинг 1).

Листинг 1. JSON сообщение для параметра “type of order”

```
{
"name" : "type of order",
"mandatory" : "true",
```

"possible values" : ["Privately", "For Business"],

"chosenValue" : ""

}

Проанализировав данный участок, можно сделать вывод, что данный параметр обязательный для выбора, а также содержит два варианта для выбора.

2) Поле для заполнения:

В данном случае мы будем писать сценарий для поля “First Name” (имя). Согласно собранным требованиям данное поле обязательное к заполнению, должно принимать только латинские буквы, от 2 до 25 знаков. В случае ввода неправильного значение поле должно подсвечиваться красным цветом и выводить текст ошибки “No valid first name entered”.

Также одно из важных требований – данное поле должно появляться, только если покупатель выберет типа заказа “Privately”.

Таким образом, данное поле будет протестировано с помощью следующих тестовых сценариев:

Нормальный случай: Пользователь вводит значение латинскими буквами от 2 до 25 знаков → ожидаемый результат: система принимает данное значение и обрабатывает его.

Альтернативный случай – заказчик выбирает “type of order” = “For business” → поле недоступно для заполнения.

Исключительный случай:

1) пользователь вводит значение, состоящее из одного символа;
2) пользователь вводит значение, состоящее из более чем 25 символов;
3) пользователь вводит значение, состоящее не из латинских символов;
4) пользователь вводит цифровое значение;
5) пользователь вводит значение, состоящее из специальных знаков.

Ожидаемым результатом для вышеперечисленных сценариев является: поле подсвечивается красным цветом и выводит текст ошибки “No valid first name entered”.

6) оставить поле пустым → ожидаемый результат: система не принимает пустое значение и просит пользователя заполнить поле.

Теперь нужно рассмотреть предложенный разработчиками шаблон сообщения (листинг 2).

Листинг 2. JSON сообщение для параметра “First Name”

```
{
"name" : "First Name",
"value" : "",
"type" : "text"
}
```

Взглянув на данную часть сообщения, можно понять, что данное поле может содержать только буквенные значения, что также подходит под указанные выше требования.

3) Дата:

В данном случае мы возьмем поле «дата рождения». Согласно собранным требованиям данное поле обязательное к заполнению, в случае если покупатель выбрал “type of order” = “Privately”. Поле «день» принимает цифровые значения от 1 до 31 или от 01 до 31, поле «месяц» принимает значения от 1 до 12 или от 01 до 12, поле «год» принимает значения от 1901 до 2002. В случае ввода неправильного значения поле должно подсвечиваться красным цветом и выводить текст ошибки “No valid first name entered”.

Также в данном случае, как и предыдущее поле, данный параметр должен появляться, только если покупатель выберет типа заказа “Privately”.

Таким образом будут протестированы следующие тестовые сценарии:

Нормальный случай: Пользователь вводит цифровые значение в обозначенных выше диапазонах –> ожидаемый результат: система принимает данное значение и обрабатывает его.

Альтернативный случай:

1) пользователь вводит дату 31/02/1999 → ожидаемый результат: не известен, так как данная ситуация не обговаривалась с заказчиком;

2) пользователь выбирает “type of order” = “For business” → ожидаемый результат: поле недоступно для заполнения.

Исключительный случай (день):

1) пользователь вводит значение менее 1;
2) пользователь вводит значение более 31;
3) пользователь вводит буквенное значение;
4) пользователь вводит значение, состоящее из специальных знаков;

Ожидаемый результат для вышеперечисленных сценариев: поле подсвечивается красным цветом и выводит текст ошибки “No valid date entered”.

5) пользователь оставляет поле пустым → ожидаемый результат: система не принимает пустое значение и подсвечивает поле красным.

Исключительный случай (месяц):

1) пользователь вводит значение 0 или 00;
2) пользователь вводит значение более 12;
3) пользователь вводит буквенное значение;
4) пользователь вводит значение, состоящее из специальных знаков

Ожидаемый результат для вышеперечисленных сценариев: поле подсвечивается красным цветом и выводит текст ошибки “No valid date entered”.

5) пользователь оставляет поле пустым → ожидаемый результат: система не принимает пустое значение и подсвечивает поле красным;

Исключительный случай (год):

1) пользователь вводит значение менее 1901;
2) пользователь вводит значение более 2002;
3) пользователь вводит буквенное значение;
4) пользователь вводит значение, состоящее из специальных знаков;

Ожидаемый результат для вышеперечисленных сценариев: поле подсвечивается красным цветом и выводит текст ошибки “No valid date entered”.

5) пользователь оставляет поле пустым → ожидаемый результат: поле должно подсвечиваться красным цветом и выводить текст ошибки “Date of birth is required”.

Помимо этого, проанализировав следующий участок сообщения, представленный в листинге 3, видно, что оно соответствует собранным ранее требованиям.

Листинг 3. JSON сообщение для параметра “Date of Birth”

```
{
"name": "Date of Birth",
"value": "",
"mandatory": "true",
"type": "masked",
"mask": "^([1-9]|0[1-9]|[12][0-9]|3[01]|)-([1-9]|0[1-9]|1[012])-(19[0-9][0-9]|200[0-2])$"
}
```

4) Выпадающий список:

В данном случае мы выбрали поле «Семейное положение» (“Family composition”). Согласно собранным требованиям, данное поле обязательное к заполнению, в случае если покупатель выберет типа заказа “Privately”, также оно не может быть пустым. Предвыбранное значение: “make a choice”. В случае, если пользователь не выберет другое значение, система не обработает заказ и попросит пользователя сделать выбор, выдав сообщение “Family composition is mandatory”.

Таким образом, данное поле можно будет протестировать следующими сценариями:

Нормальный случай: Пользователь выбирает одно из предложенных значений → ожидаемый результат: система принимает данное значение и обрабатывает его.

Альтернативный случай:

1) пользователь выбирает “type of order” = “For business” → ожидаемый результат: поле недоступно для заполнения.
2) пользователь пробует ввести значение вручную с клавиатуры → ожидаемый результат: система не дает такой возможности.

Исключительный случай: Пользователь не делает выбор, оставляя предвыбранное системой значение (“make a choice”) → ожидаемый результат: система не обрабатывает заказ и выводит сообщение “Family composition is mandatory”.

Часть сообщения, относящаяся к данному полю, также соответствует вышеперечисленным требованиям (листинг 4).

Листинг 4. JSON сообщение для параметра “Family composition”

```
{
"name" : "Family Composition",
"possibleValues" : ["make a choice", "Single", "Single with children",
"Living together/ married", "Living together/ married with children"],
"chosenValue" : "",
"mandatory" : "true",
"type" : "list"
}
```

5) Поле содержащее регулярное выражение:

В данном случае мы выбрали параметр “Account no. (IBAN)”, т.е. номер банковского счета. Согласно собранным требованиям данное поле обязательное к заполнению и не может быть пустым. Регулярное выражение, использующееся в данном случае:

NL[0-9]{2}\\s[A-Z]{4}\\s[0-9]{4}\\s[0-9]{4}\\s[0-9]{2}

В случае, если пользователь оставит поле пустым, он увидит сообщение “Account number is required”, в то время как если он введет значение, неподходящее под регулярное выражение, поле должно подсвечиваться красным и выдавать ошибку “No valid account number entered”.

Таким образом, тестирование данного поля будет проходить по следующим сценариям:

Нормальный случай: Пользователь вводит значение, подходящее под регулярное выражение → ожидаемый результат: система принимает данное значение и обрабатывает его.

Альтернативный случай: отсутствует

Исключительный случай:

1) Пользователь вводит значение, неподходящее под регулярное выражение → ожидаемый результат: система не обрабатывает заказ и выводит сообщение “No valid account number entered”;
2) Пользователь оставляет поле пустым → ожидаемый результат: система выводит сообщение “Account number is required”;

Теперь давайте рассмотрим участок сообщения для данного поля, представленный в листинге 5.

Листинг 5. JSON сообщение для параметра “Account no. (IBAN)”

```
{
"name" : "Account no. (IBAN)",
"value" : "",
"mandatory" : "true",
"type" : "masked",
"mask" : "NL[0-9]{2}\\s[A-Z]{4}\\s[0-9]{4}\\s[0-9]{4}\\s[0-9]{2}"
}
```

Проанализировав предоставленную часть, тестировщик может сделать вывод, что данное поле обязательно для заполнения, и имеет свою «маску», под которую должно подходить введенное значение, что соответствует ранее собранным требованиям.

Таким образом, после сбора и анализа требований тестировщик может составить тестовые сценарии, используя все возможные методы, о которых он знает, в данном случае мы использовали методологию случаев использования, также классы эквивалентности, анализ пограничных значений, а также анализ

покрытия кода. Проанализировав составленные сценарии, можно сделать вывод, что они соответствуют главному требованию, т.е. покрывают все основные функции в требованиях.

3.2.2 Интеграционное тестирование, отслеживание найденных ошибок

В данном случае мы будем тестировать интеграцию первых двух систем: первая – где пользователь вводит свои данные, вторая – система для дальнейшей обработки заказа. Для интеграционного тестирования обычно используется логирование, которое в данном случае мы не можем показать, однако мы можем предоставить примерное описание, как оно выглядят.

Итак, каждый заказ имеет свой уникальный идентификатор, по которому можно найти все сообщения, принадлежащие данному заказу. Помимо этого, каждое сообщение, кроме самого текста, должно содержать заголовки (headers), которые несут информативную цель, часто они содержат уникальный идентификатор заказа и заказчика, временные штампы, хост, с которого и на который было направленно сообщение, медиа тип (в данном случае *application/json*) и т.п. Не стоит пренебрегать тестированием данной части, так как, если в ней указана неверная информация, это может привести к ошибкам. Помимо этого, часто в логах показаны HTTP коды состояния, которые отображают успешность или не успешность той или иной операции.

Итак, при интеграционном тестировании проверяется взаимодействие двух систем. В данном случае, для того чтобы начать тестирование, нам нужно зайти на официальный сайт телеком компании и выбрать телефон, который мы хотим купить (рис. 3.1.).

В данном случае мы выбираем первый телефон – Apple iPhone12 65GB, но также мы видим, что покупатель может выбрать любой телефон и при повторных тестовых прогонах можно выбрать другой телефон, для того чтобы убедиться, что при любом раскладе система работает одинаково.

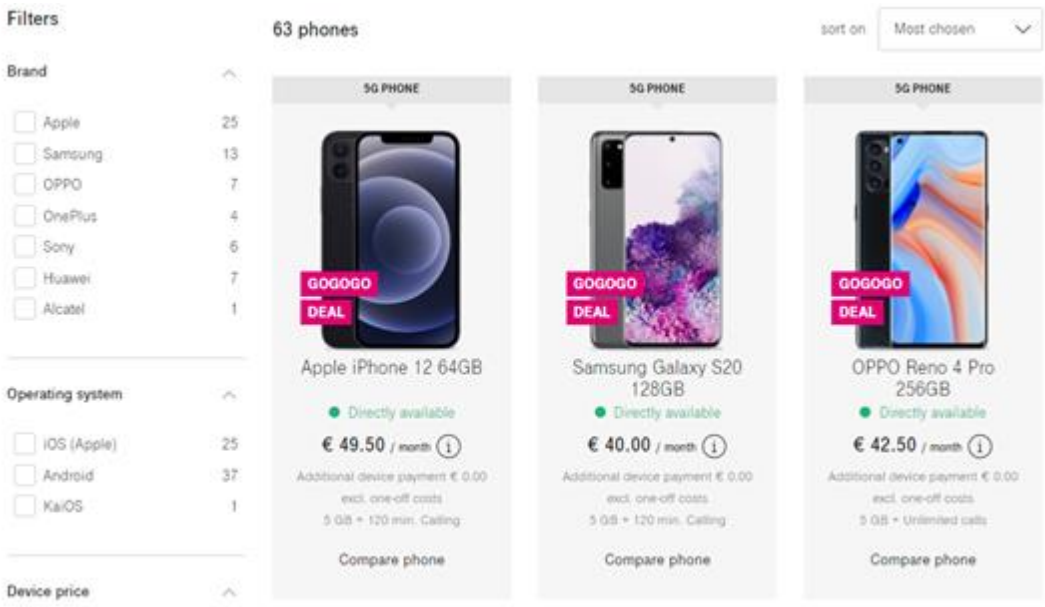


Рисунок 3.1 – Стартовая страница для выбора телефона

После выбора телефона мы видим, что прежде чем перейти на форму заполнения, которая была подробно описана в предыдущем пункте, покупателю нужно выбрать тариф (рис. 3.2.), для которого сразу калькулируется сумма. Сразу нужно отметить, что данная сумма и должна передаться в последующие системы, чтобы, когда покупатель получил чек на оплату, у него была такая же сумма.

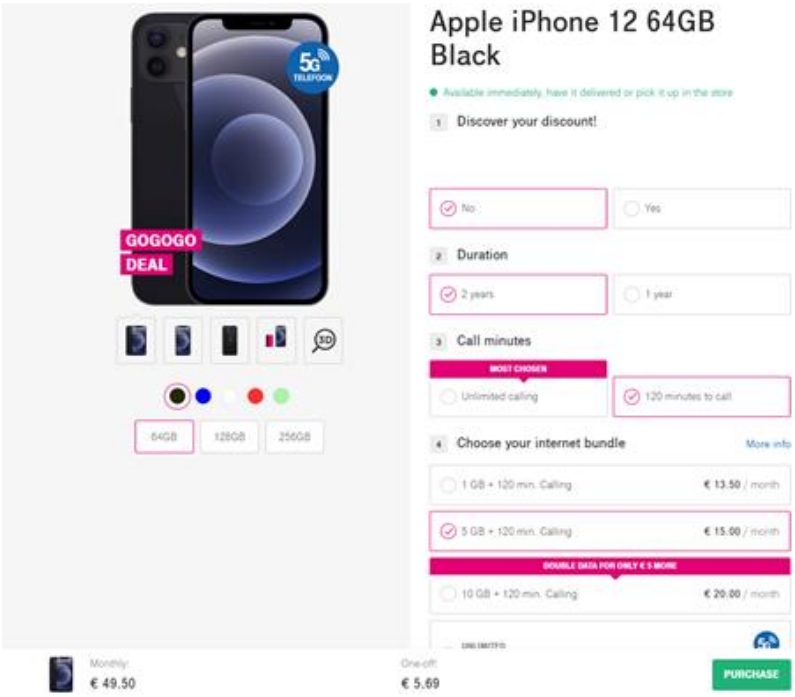


Рисунок 3.2 – Выбор тарифа

После этого мы можем начать заполнение формы, часть которой представлена в приложении 2. Итак, мы видим, что форма содержит все поля, которые запрашивал заказчик. На данном этапе мы возвращаемся к составленным ранее тестовым сценариям и начинаем непосредственно их проведение. Как можно увидеть в предыдущем пункте, все тестовые сценарии разделены на три группы – нормальный, альтернативный и исключительный случаи. В данном пункте мы объединим все нормальные случаи в один тестовый прогон, чтобы сократить время тестирования.

Итак, после заполнения данной формы, мы должны увидеть следующую картину:

1) “type of order” = “Privately”;
2) присутствует секция “My details”, в которую входят поля:
    a. “First Name” – пользователь вводит значение Iuliia;
    b. “Date of Birth” – пользователь вводит значение 20-04-1996;
3) присутствует секция “Income and burden test” (Информация о доходах и расходах), в которую входит выпадающий список «Family composition» – пользователь выбирает одно из значений выпадающего списка (например, Single);
4) присутствует секция “ID”, в которую входит параметр “Account no. (IBAN)”, пользователь вводит подходящее значение (например, NL79 INGB 5138 7731 27);

В приложении 2 видно, что при выбранном “type of order” = “Privately”, все выше перечисленные параметры и секции присутствуют.

После введения всей нужной информации, указанной в тестовом сценарии, система принимает значения и отправляет заказ на обработку, следовательно, значения из сценария «нормальный случай» приняты, поэтому можно считать его пройденным.

Помимо этого, так как интеграционное тестирование подразумевает общение между двумя и более системами, в данном случае нам нужно обратиться к логированию, чтобы убедиться, что сообщение, отправленное в

следующую систему правильное, для этого мы и анализировали шаблон JSON сообщения в предыдущем пункте.

Итак, посмотрев на сообщение, можно увидеть, что, реальное сообщение совпадает с шаблоном, а также что следующие параметры заполнены корректно:

1) параметр “type of order” (листинг 6):

Листинг 6. Заполненное JSON сообщение для параметра “type of order”

```
{
"name" : "type of order",
"mandatory" : "true",
"possible values" : ["Privately", "For Business"],
"chosenValue" : "Privately",
}
```

2) параметр “First Name” (листинг 7):

Листинг 7. Заполненное JSON сообщение для параметра “First Name”

```
{
"name" : "First Name",
"value" : "Iuliia",
"type" : "text"
}
```

3) параметр “Date of Birth” (листинг 8):

Листинг 8. Заполненное JSON сообщение для параметра “Date of Birth”

```
{
"name": "Date of Birth",
"value": "20-04-1996",
"mandatory": "true",
"type": "masked",
"mask": "^([1-9]|0[1-9]|[12][0-9]|3[01]|)-([1-9]|0[1-9]|1[012])-(19[0-9][0-9]|200[0-2])$"
}
```

4) параметр “Family composition” (листинг 9):

Листинг 9. Заполненное JSON сообщение для параметра “Family composition”

{
"name" : "Family Composition",
"possibleValues" : ["make a choice", "Single", "Single with children", "Living together/ married", "Living together/ married with children"],
"chosenValue" : "Single",
"mandatory" : "true",
"type" : "list"
}

5) параметр “Account no. (IBAN)” (листинг 10):

Листинг 10. Заполненное JSON сообщение для параметра “Account no. (IBAN)”

{
"name" : "Account no. (IBAN)",
"value" : "NL79 INGB 5138 7731 27",
"mandatory" : "true",
"type" : "masked",
"mask" : "NL[0-9]{2}\\s[A-Z]{4}\\s[0-9]{4}\\s[0-9]{4}\\s[0-9]{2}"
}

Согласно логам, система ответила кодом 202 – Accepted, следовательно, система, принимающая данный запрос, приняла его и начала дальнейшую обработку.

Также согласно бизнес-требованию, после заполнения данной формы пользователь должен увидеть номер своего заказа. Который поступил в обработку. Номер заказа в данном случае возвращает та система, которая начала обработку, что видно на рис. 3.3.

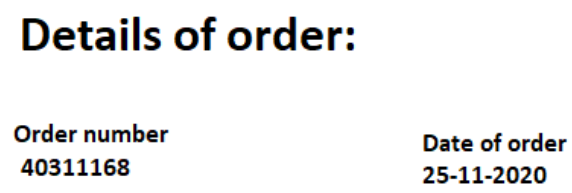


Рисунок 3.3 – Номер заказа

Согласно логированию, вернувшееся сообщение выглядит следующим образом (листинг 11).

Листинг 11. JSON сообщение с вернувшимся номером заказа

```
{
"name" : "Details of order",
"value" : [
{
"name" : "Order number",
"value" : "40311168"
},
{
"name" : "Date of order",
"value" : "25-11-2020"
}
]
}
```

Следующий тестовый случай – альтернативный. Согласно списку составленных тестовых сценариев, наш тестовый прогон будет содержать следующие параметры:

1) “type of order” = “Business”;
2) отсутствует секция “My details”;
3) отсутствует секция “Income and burden test”;
4) присутствует секция “ID”: так как альтернативный случай для данного параметра отсутствует мы введем значение для нормального случая, например, NL75 ABNA 9372 7183 00.

В приложении 3 видно, что при выбранном “type of order” = “For Business”, отсутствуют все выше перечисленные секции (кроме ID), что и является ожидаемым результатом.

После введения всей информации, указанной в тестовом сценарии, система принимает значения и отправляет заказ на обработку, также проанализировав отправленное сообщение, можно сделать вывод, что реальное сообщение совпадает с шаблоном, а также что тестовый сценарий можно считать пройденным:

1) параметр “type of order” (листинг 12):

Листинг 12. Заполненное JSON сообщение для параметра “type of order”

```
{
"name" : "type of order",
"mandatory" : "true",
"possible values" : ["Privately", "For Business"],
"chosenValue" : "For Business"
}
```

2) параметр “First Name” – отсутствует, следовательно, данной части в сообщении нет.
3) параметр “Date of Birth” – отсутствует;
4) параметр “Family composition” – отсутствует;
5) параметр “Account no. (IBAN)" – параметр принят, сообщение для нормального случая показано в листинге 10.

Согласно логам, система ответила кодом 202 – Accepted, следовательно, система запрос принят взят в обработку. Также после заполнения всей формы мы можем увидеть номер и дату заказа, которые выглядят также, как и для “type of order” = “Privately” (рис. 3.3)

Следующий тестовый случай – исключение, содержит больше всего тестовых сценариев, однако в данной работе мы рассмотрим только один из них, но на практике в проверке должны учувствовать все тестовые сценарии.

Итак, наш тестовый прогон будет содержать следующие шаги:

1) Параметр “type of order” – пользователь оставляет данное поле пустым;
2) Параметр “First Name” – пользователь оставит данный параметр незаполненным;
3) Параметр “Date of Birth” – пользователь вводит невалидное значение, например, 33-50-2023;
4) Параметр “Family composition” – пользователь оставляет предвыбранное значение “make a choice”;
5) Параметр “Account no. (IBAN)” – пользователь вводит не валидное значение, например, NL00 0000 0000 0000.

Согласно ожидаемому результату система не должна принять ни одно из вышеперечисленных значений и выдать ошибку:

1) Тестовый случай пройден – система не дает оставить данное поле пустым, одно значение всегда должно быть выбрано;
2) Тестовый случай *НЕ* пройден, так как система принимает пустое значение для данного параметра;
3) Тестовый случай пройден – система не дает ввести невалидное значение, поле подсвечиваются красным и выводиться текст ошибки (рис. 3.4);

Рисунок 3.4. – система не принимает не валидное значение.

4) Тестовый случай пройден, система выдает ошибку (рис. 3.5.);

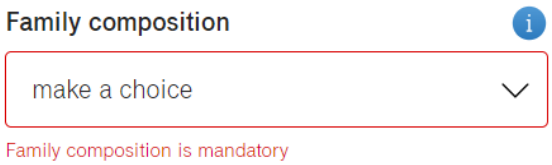


Рисунок 3.5. – система не принимает не валидное значение

5) Тестовый случай пройден – система не дает ввести невалидное значение (рис 3.6.);

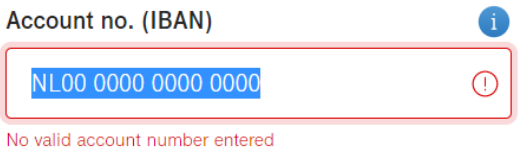


Рисунок 3.5. – система не принимает не валидное значение

Таким образом, протестировав все тестовые сценарии, была обнаружена ошибка, связанная с параметром “First Name”. В данном случае тестировщику необходимо зафиксировать ее в системе отслеживания ошибок, чтобы разработчики ее увидели и исправили.

В качестве примера мы будем использовать бесплатную систему отслеживания ошибок – JiraSoftware.

Для корректного создания ошибки нужно описать то, как ошибка была найдена (воспроизведена), описать ожидаемый результат и реальный. Можно приложить причину данной ошибки, если она известна, в данном случае это, например, отсутствие в JSON шаблоне такого параметра как “mandatory”, который определяет обязательность поля. Если у тестировщика есть дополнительная информация, ее также нужно приложить, в нашем случае это логирование.

Помимо этого, тестировщику необходимо определить приоритет данной задачи, так как существуют и низкоприоритетные ошибки, которые могут быть исправлены во вторую очередь. В нашем случае мы определили, что приоритет данной задачи средний.

Также тестировщик может оставить метки, по которым можно будет настроить отдельные фильтры в системе или производить поиск по значению данных меток. Кроме того, нужно назначить исполнителя (приложение 4).

После того как ошибка взята в работу тестировщик должен следить за ее исполнением, если ошибка высокого приоритета, нужно добиться того, чтобы она была исправлена как можно скорее.

После того, как разработчики исправили данную ошибку, тестировщик должен повторно протестировать сценарий и убедиться в том, что ожидаемое поведение соответствуют реальному.

Таким образом, нами было проведено интеграционное тестирование для официального сайта одной телеком компании. Нами были протестированы составленные ранее тестовые случаи и сценарии, в результате чего была найдена и зафиксирована ошибка, связанная с валидациями.

3.2.3 Тестирование пользовательского интерфейса, отслеживание найденных ошибок

Во второй главе данной работы были описаны критерии, применимые к данному виду тестирования, они также применимы и к нашей системе. Следовательно, после проведения тестирования графического интерфейса мы сделали следующие выводы:

1) вид элементов при уменьшении окна браузера остается приемлемым, ни один элемент не «съезжает» или не исчезает;
2) весь текст написан правильно, отсутствие грамматических ошибок;
3) присутствует возможность перемещения фокуса с использованием клавиш Tab или Tab+Shift;
4) присутствует возможность использования горячих клавиш;
5) присутствует возможность выделения отдельных элементов/ текста;
6) дизайн единообразен;
7) возможность переключать и нажимать на кнопки, возможность выбрать тот или иной чек-бокс;
8) возможность выбрать значение из выпадающего списка;
9) всплывающие окна не раздражают пользователя, есть возможность их скрытия
10) присутствие всех нужных информационных боксов.

Однако и не обошлось без минусов, например, изначально пользователь не видит того, какие параметры являются обязательным, поэтому, если он не заполнит какой-либо из обязательных параметров, он увидит это только после того, как страница перезагрузится и подсветит незаполненные поля красным. В

данном случае это можно считать незначительной ошибкой, так как важнее, чтобы обязательные поля были определены в коде.

Таким образом, тестирование графического интерфейса системы было проведено успешно, была найдена одна незначительная ошибка, которая была занесена в систему отслеживания.

3.2.4 Системное тестирование, отслеживание найденных ошибок

Как нам уже известно, системное тестирование – это расширенное интеграционное тестирование, с отличием в том, что тестируется интеграция не двух отдельных компонент, а работа системы в целом.

В данном случае, нам нужно протестировать всю цепочку действий, описанных в пункте 3.2.

На данном этапе нам нужно протестировать весь процесс целиком, то есть получается настоящее end-to-end тестирование. К данному моменту все ошибки, найденные на предыдущих этапах должны быть исправлены и протестированы. А также помимо этого должны быть проведены отдельные интеграционные тесты для проверки платежеспособности, получения чека и произведения оплаты.

Тестировщик также, как и при интеграционном тестировании, может смотреть логирование, но в то же время, так как подробное логирование тестировалось на интеграционном уровне, подробное рассмотрение логов можно опустить, только если не будет найдена ошибка.

Так как за проверку платежеспособности отвечает система другого вендора, у тестировщика есть несколько вариантов тестирования данного шага:

1) Сторонний вендор также настраивает тестовое окружение, которое связано с тестовым окружением тестировщика. На данном тестовом окружении можно настроить автоматические ответы для желаемого результата, например, «Принят» (платежеспособность подходит для покупки данного товара) или «Отклонен» (платежеспособность менее ожидаемого);

2) Тестировщик с использованием специальных тестовых инструментов сам отправляет нужные сообщения.

В данном случае первый вариант предпочтительнее, так как он эмулирует реальное поведение без каких-либо «заглушек», что и требуется достичь на данном шаге тестирования.

Следовательно, тестировщик снова должен пойти на главную страницу сайта проделать предыдущие шаги начала процесса, затем проверить что система компании «НетКрэкер» и система, отвечающая за проверку платежеспособности, также успешно проинтегрировали, в результате чего был создан документ на оплату.

Затем тестировщик должен проверить, что у тестового покупателя в личном кабинете была сформирована ссылка на оплату. Пройдя по данной ссылке, пользователь должен увидеть ту же информацию об оплате, которую он видел в самом начале выбор телефона, а также реквизиты для оплаты.

После того как банк подтвердил перевод средств, пользователь должен увидеть информацию о доставке или информацию, когда и где он может забрать свой заказ.

Выше был приведен пример позитивного системного тестирования, однако бывают и негативные сценарии. Например, когда платежеспособность была отклонена или пользователь в последний момент перед оплатой решил отменить заказ. Все данные сценарии должны быть продуманы и протестированы в полной цепочке от начала и до конца.

При нахождении ошибок тестировщики должны зафиксировать их в системе отслеживания и, также, как и любом этапе тестирования, отслеживать их исправление.

Как только ошибки будут исправлены, тестировщики должны пройти всю цепочку от начала до конца заново, чтобы убедиться, что нужный функционал исправлен, а тот, который работал, все еще работает.

Таким образом, системное тестирование является расширенной версией интеграционного тестирования, однако на данном шаге должна быть

протестирована вся цепочка от начала до конца. В данном пункте мы привели примеры таких цепочек согласно тестируемой программы.

3.2.5 Регрессионное тестирование

Во-первых, в данном пункте стоит отметить, что регрессионное тестирование является не только последним шагом, оно может являться и промежуточным, когда разработчик исправил ту или иную ошибку, или, когда был добавлен новый функционал, регрессионное тестирование поможет удостовериться в том, что старый функционал не сломан.

Однако в данном случае мы используем данный шаг как подготовку к выдаче. С помощью регрессионного тестирования тестировщик в последний раз должен убедиться, что протестированная система работает согласно требованиям, вся документация обновлена до актуального состояния, а все высокоприоритетные ошибки исправлены.

Применимо к данному примеру, задача тестировщика – провести системное тестирование еще раз (включая позитивные и негативные сценарии), однако без детального изучения логирования и отправляемых/ получаемых сообщений. В данном случае можно использовать метод черного ящика, т.е. смотреть на систему чисто со стороны пользователя, который не знает ни о каких требования, у него есть одна цель – сделать заказ, оплатить и получить его.

С течением проекта, когда функционал становится сложнее, самым оптимальным методом проведения регрессионного тестирования является автоматизация. Автоматизация значительно сокращает время тестирования сценариев, особенно, если в основной функционал больше не вносится изменений, а основной акцент делается на разработку нового.

Однако, стоит отметить, что на автоматизацию уходит много времени, особенно для тестирования сложных сценариев, включающих валидацию сообщений, а также потому что автоматизация требует постоянного обновления, тем более для регрессионных целей. Помимо этого, специалист,

разрабатывающий такие тесты должен обладать знанием языков программирования (по крайней мере одного), поэтому часто на проектах создается две команды тестирования. Первая команда разрабатывает автоматические тесты, вторая – использует созданный материал, а также тестирует новый функционал (часто вручную).

Таким образом, регрессионное тестирование для данной системы было проведено успешно, новых ошибок не выявлено, после чего можно сделать вывод, что продукт готов к выдаче. Также для целей регрессионного тестирования часто создается автоматизированные тесты, что сокращает время проведения тестирования, однако не стоит забывать, что на разработку и поддержку таких тестов уходит много времени, а специалист, разрабатывающих их, должен обладать как минимум одним языком программирования.

3.2.6 Выдача

На данном этапе задача тестировщика – написать руководство по установке сборки на реальный сервер, а также написать руководство по настройки каких-либо новый функций. Данный шаг ни в коем случае нельзя пропускать, так как без правильных настроек некоторые функции системы могут работать некорректно.

В данном случае на текущем проекте, помимо руководства по установке, должно быть написано только одно руководство: «Как включить логирование для новых сообщений», так как в случае каких-либо ошибок будет невозможно посмотреть отправленные/ полученные сообщения, что приведет к дальнейшим проблемам.

Руководство пользования содержит конкретные шаги. К сожалению, мы не можем показать реальное руководство пользования, однако мы можем привести пример, как оно могло бы выглядеть:

1) зайти на реальный сервер;

2) перейти на страницу «*название страницы*», где находятся включатели и выключатели для всех существующих запросов, проходящих через систему «НетКрэкер»;
3) найти название запроса на отправку сообщения – «*название*»;
4) перевести его в состояние «*вкл*»;
5) найти название запроса на отправку чека для оплаты – «*название*»;
6) перевести его в состояние «*вкл*»;

Руководство по установке содержит следующие пункты:

1) предварительные условия:

– На сервере, куда собираются устанавливать новую сборку, уже должна стоять предыдущая сборка, в противном случае установить предыдущую.

2) шаги для установки сборки:
   a. Запустить специальную программу для установки;
   b. Взять подготовленную сборку (обычно это .zip файлы);
   c. Используя специальную программу, распаковать файл специальной командой;
   d. Ввести команду для установки сборки на сервер.

Таким образом, на данном шаге нами были написаны руководства по установке и настройки некоторых функций на сервере. Данные руководства помогут людям, которые работают непосредственно с реальными серверами сэкономить время и не разбираться досконально в новом функционале, для них достаточно будет следовать уже составленной инструкции.

Выводы по третей главе

В данной главе была представлена и протестирована новая методика тестирования, которая систематизирует разрозненные данные о тестировании программного обеспечения. Данная методика состоит из следующих шагов: сбор требований о тестируемой системе, анализ требований, разработка тестовых сценариев, интеграционное тестирование, отслеживание найденных ошибок, тестирование пользовательского интерфейса и отслеживание

найденных ошибок, системное тестирование, отслеживание новых ошибок, тестирование пользовательского интерфейса, регрессионное тестирование и выдача проекта.

На шаге сбора требований мы представили систему, которую собираемся тестировать, а также собрали требования заказчика для нового функционала. На следующем шаге требования были проанализированы с точки зрения адекватности, полноты, совместимости, выполнимости и разумности, а также с точки зрения подверженности тестированию. После обсуждения и доработки требований все изменения были внесены в техническую документацию. На шаге разработки тестовых сценариев мы составляли примеры сценариев для стартовой страницы одного телеком оператора, были взяты типовые поля шаблона, который нам предстояло протестировать, и составлены тестовые сценарии с помощью методологии случаев использования, техники классов эквивалентности и анализа пограничных значений, а также анализа покрытия кода. На шаге интеграционного тестирования мы протестировали интеграцию двух систем, первая система принимала данные от пользователя, в то время как вторая принимала их для дальнейшей обработки. Были подробно протестированы составленные ранее тестовые сценарии, в результате чего была найдена и зафиксирована ошибка. Затем пользовательский интерфейс был протестирован с точки зрения различных критериев, после чего был найден незначительный недочёт. На шаге системного тестирования были проверены различные позитивные и негативные end-to-end процессы.

Регрессионное тестирование было проведено успешно и новых ошибок обнаружено не было. Последним шагом явилась выдача продукта, на которой были составлены руководство по настройке новых функций, а также руководство по установки выданной сборки на реальный сервер.

Тестирование программы было проведено в основном с помощью метода «серого ящика», без доступа к исходному коду, но с доступом к техническому заданию и логировнию.

Глава 4 Рекомендации по использованию представленной методики тестирования

4.1 Обоснование эффективности представленной методики тестирования

Данная методика была внедрена в проект компании «НетКрэкер» в мае 2020 года в одной команде, а затем в ноябре 2020 года внедрена во второй. В данном пункте будет рассмотрена эффективность данной методики, ее достоинства и недостатки на примере первой команды, которая работает по данной методике уже более двадцати спринтов. Вторая команда только начала свой переход на данную методику, и еще рано судить о результатах.

Стоит отметить, что данный проект работает по гибкой методологии Scum, где команда выполняет свою работу в течение определенного временного интервала, который называется спринт. На данном проекте спринт длится две недели, а выдача разработанного функционала происходит один раз в месяц (после каждого второго спринта). Руководство проекта со стороны заказчика постоянно оценивает эффективность работы команд, поэтому каждый спринт создается отчетность о разработанном функционале, на основе которой создаются разные графики для различных пунктов оценивания команд.

Изначально команда испытывала трудности с переходом на данную методику, так как она затрагивает не только тестировщиков. Согласно новой методике, модульное тестирование (с использованием заглушек) проводится разработчиками, а к тестировщикам уже приходит отчасти протестированный функционал, благодаря чему количество ошибок, найденных тестировщиками при интеграционном тестировании стало уменьшаться.

Согласно первому графику (рис. 4.1), где ось Y – это количество найденных ошибок, а ось X – течение времени, видно, что до и в начале перехода на новую методику тестировщики находили бóльшее количество ошибок, некоторые из которых были критическими или блокирующими. После перехода на новую методику количество ошибок, в частности блокирующих, сократилось до минимума, тем самым, команда начала изначально поставлять более качественный продукт. Это произошло благодаря более качественно

проведенным дев-тестированиям. Разработчики начали сами находить и исправлять ошибки до выдачи их тестировщикам. Также стоит отметить, что разработчики начали «меняться» функционалом, т.е. один разработчик пишет код, второй тестирует.

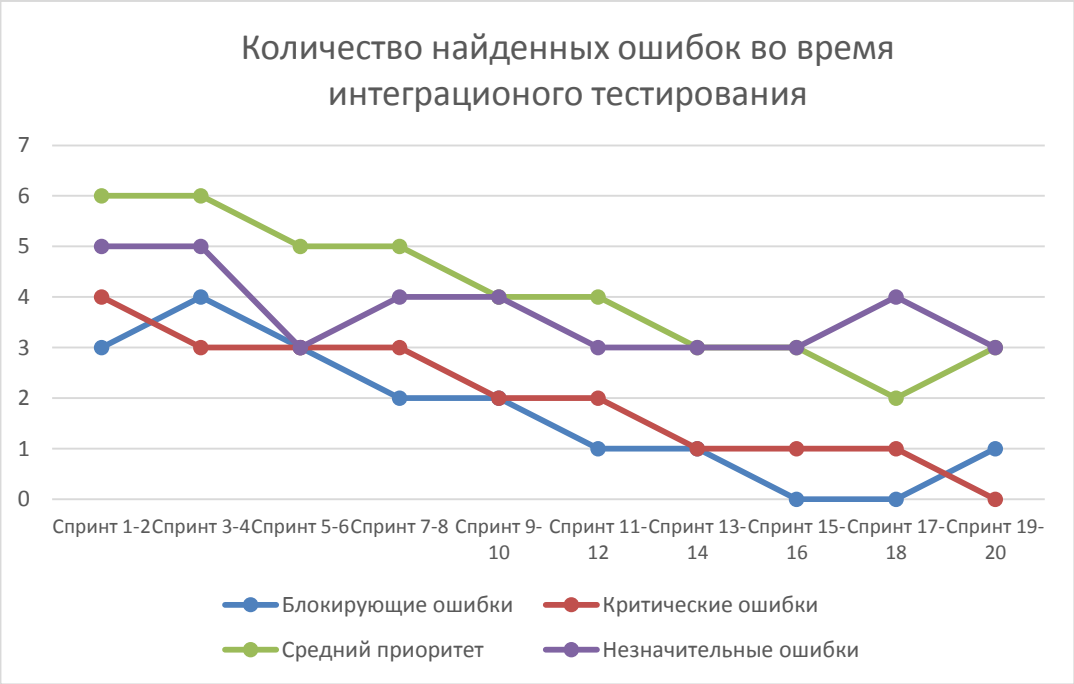


Рис. 4.1 – Количество найденных ошибок во время интеграционного тестирования

Следующий график (рис.4.2) тесно связан с предыдущим, т.к. он отражает процент непройденных тестовых сценариев после первого прогона интеграционных тестов. Данный процент начал снижаться после перехода на описанную в предыдущей главе методику тестирования.

Следующий график (рис. 4.3) показывает примерное время, затраченное на прохождение одного тестового сценария на этапе системного тестирования, т.е. с того момента, как тестировщик взял сценарий в непосредственную работу и до момента его закрытия со статусом «пройден». Это время включает в себя нахождение всех ошибок и их повторное тестирование. График показывает, что со снижением количества ошибок, время, затраченное на прохождение одно сценария, уменьшилось, тем самым у тестировщика остается больше времени на проверку других сценариев, а также составление более сложных сценариев с различными вводными значениями.

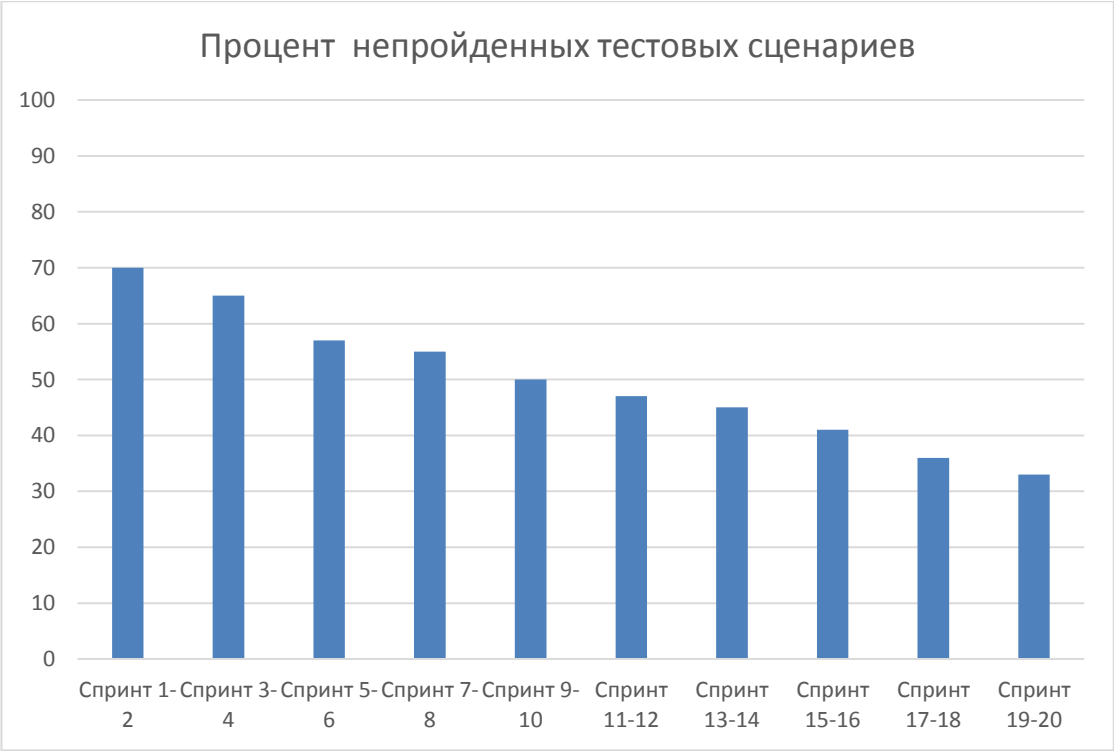


Рис. 4.2 – Процент не пройденных тестовых сценариев

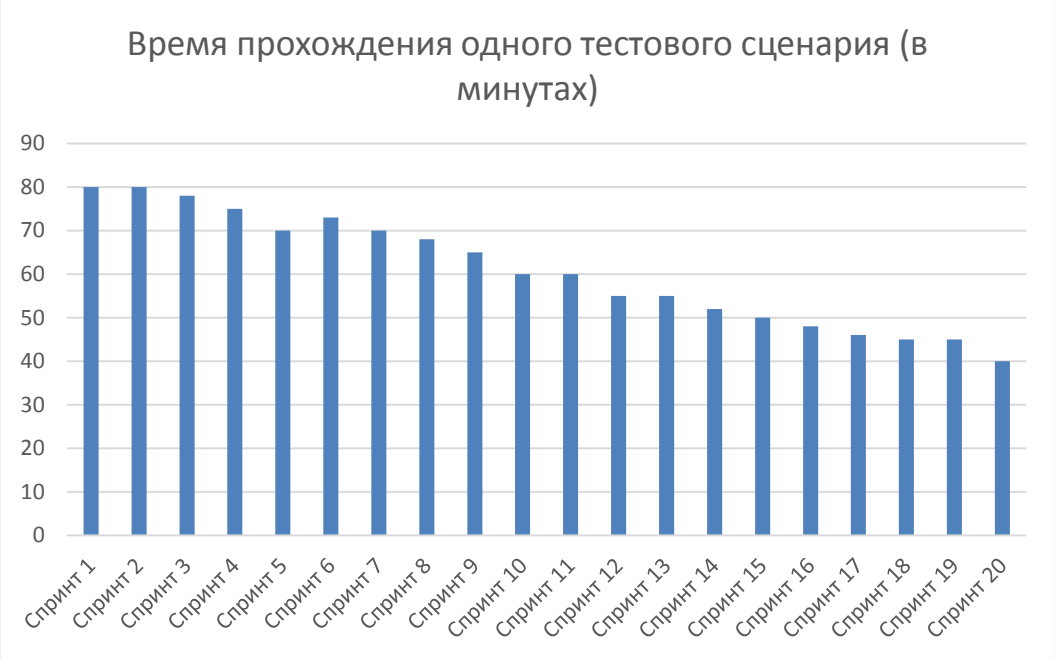


Рис. 4.3 – Время прохождения одного тестового сценария на этапе системного тестирования

Следующий пункт оценки эффективности – загруженность тестировщиков во время спринта. Согласно графику (рис. 4.4) видно, что до и

вначале перехода на новую методику команда тестировщиков была перегружена из-за проведения всех видов тестирования, на что уходило много времени. На графике видно, что со временем эта нагрузка уменьшилась и нормализовалась, благодаря тому, что разработчики забрали на себя модульное тестирование.

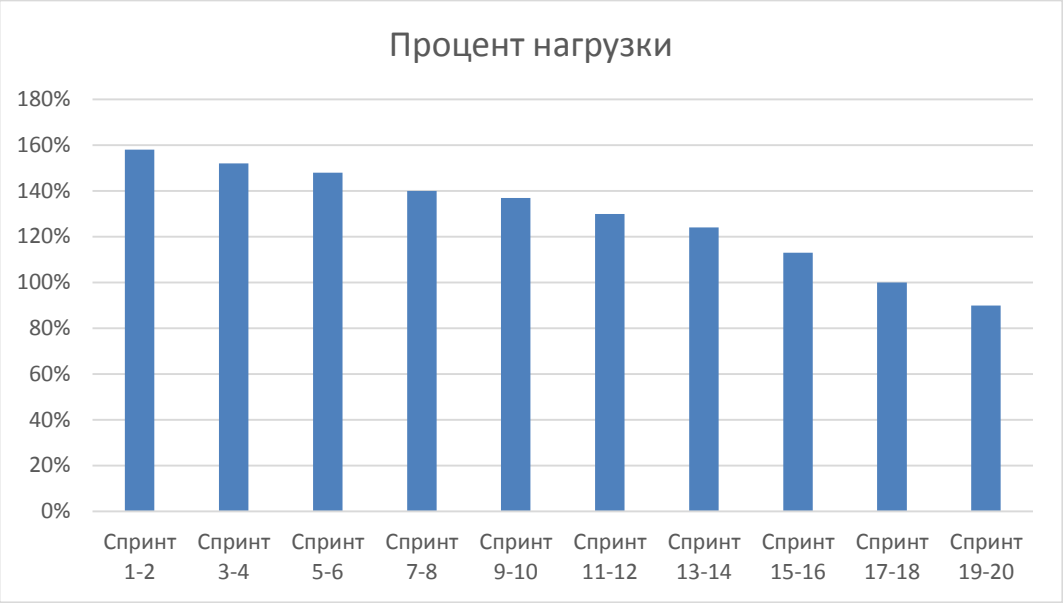


Рис. 4.4 – Процент нагрузки на команду тестировщиков

Также стоит отметить, что в гибких методологиях разработки ПО присутствует такое понятие, как «пользовательская история», которая отражает одну функциональную возможность, описанную с точки зрения конечного пользователя (например, возможность оплаты заказа). Следующий график показывает, что с переходом на описанную методику, количество пользовательских историй, выдаваемых заказчику уменьшилось. Это произошло из-за перевода разработчиков на модульное тестирование, что повысило их нагрузку в течение одного спринта. Для заказчика это является несомненным недостатком, даже несмотря на то, что поставляемый функционал повысил свое качество. Помимо этого, следующий график показывает количество историй

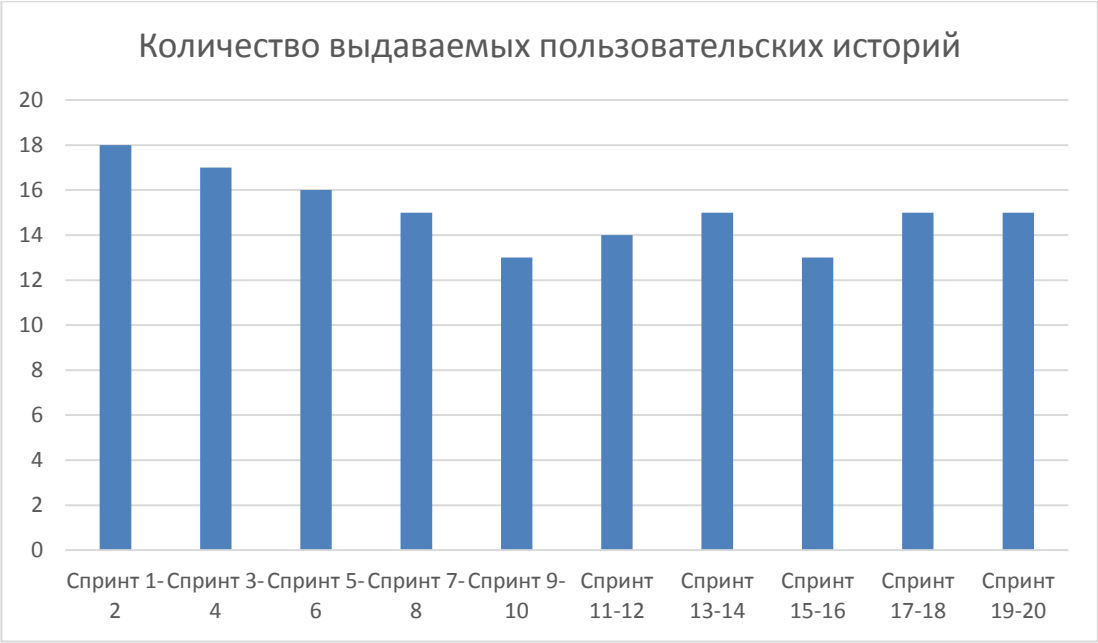


Рис. 4.5 – Количество выдаваемых пользовательских историй

После анализа вышеперечисленных оценок эффективности можно выделить достоинства данной методики.

Во-первых, она систематизирует существующие виды тестирования в одну понятную схему с конкретными шагами (приложение 1). Это полезно, как и для начинающих, так и для опытных тестировщиков.

Во-вторых, тестировщики перестают проверять один и тот же функционал по несколько раз на разных уровнях, это значительно сокращает время на тестирование и снижает нагрузку на команду тестировщиков.

В-третьих, данная методика повышает качество поставляемого функционала за счет более качественно проведенного тестирования.

В-четвертых, данная методика основана на методе «серого ящика», что позволяет протестировать бóльшее количество тестовых сценариев, чем в методе «черного ящика», а также она является менее трудозатратной, чем метод «белого ящика».

Однако данные достоинства не исключают своих недостатков. Во-первых, данная методика в основном подходит под гибкие методологии, так как она подразумевает частое общение с командой разработчиков из-за их вовлеченности в процесс модульного тестирования. Помимо этого, вся команда

разработчиков (а не только ведущий разработчик) должна быть вовлечена в бизнес-процесс разрабатываемого продукта, чего практически не существует в каскадных методологиях. Во-вторых, данная методика, как показывает практика, снижает количество поставляемого функционала из-за возрастания нагрузки на команду разработчиков.

Помимо этого, в ходе тестирования данной методики на практике, стало ясно, что данная методика может быть усовершенствована тем, чтобы начать вовлекать разработчиков не только в модульное тестирование, но и в интеграционное. Это может еще больше повысить качество поставляемого функционала.

Данный переход уже имеет место быть в первой команде, которая уже долгое время работает по данной методике. Команда разработчиков выбирает наименее сложный функционал, взятый в спринт, обсуждает его, анализирует с точки зрения его интеграции с другим функционалом/ системами, составляет тестовые сценарии иногда совместно с командой тестировщиков, а затем выполняет данные тесты. Однако, пока что рано судить об эффективности этого перехода.

Таким образом, данная методика была внедрена в один из проектов компании «НетКрэкер» в двух командах. На примере одной из команд удалось проанализировать эффективность данной методики, в результате чего стало видно, что команда разработчиков стала более вовлеченной в бизнес-процесс проекта, а благодаря более тщательному модульному тестированию, в интеграционное тестирование стало приходить меньшее количество критических ошибок, также стоит отметить, что благодаря данной методике снизилась нагрузка на команду тестировщиков. В то же время недостатком данной методики является уменьшение количества поставляемого функционала.

4.2 Возможности применения результатов представленной методики тестирования

Проанализировав данные описанные в предыдущих главах, можно сделать вывод о возможностях применения составленной методики тестирования.

Во-первых, данная методика является достаточно универсальной (особенно для проектов, работающих по гибкой методологии) и может быть применена на разных проектах разработки ПО, например, сайты (в особенности те, которые интегрируют с другими системами), фронтэнд и бэкэнд приложения, базы данных, и т.п.

Во-вторых, так как данная методика дает конкретные шаги, по которым должно проводиться тестирование ПО, она дает возможность начинающим тестировщикам быстрее «влиться» в процесс тестирования.

В-третьих, данная методика подойдет IT проектам, которые хотят повысить качество выдаваемого функционала, или проектам, в которых слишком большой процент переоткрытых ошибок.

Таким образом, новая методика тестирования систематизирует наиболее популярные методы и техники тестирования, а также дает представления об оптимальном процессе тестирования. Методика может применяться для тестирования разных IT-проектов, например, web-сайты, бэкэнд приложения и т.п., а также подходит под проекты, которые хотят увеличить качество разрабатываемого продукта.

Выводы по четвертой главе

Таким образом, новая методика тестирования структурирует наиболее популярные методы и техники тестирования, а также дает представление о том, что наиболее оптимальным методом тестирования является метод «серого ящика». Методика может применяться для тестирования разных IT-проектов, например, web-сайты, фронтэнд и бэкэнд приложения и т.п., но больше подходит для проектов, работающих по гибкой методологии (например, Scrum),

или на проектах, которые стремятся улучшить качество разрабатываемого функционала.

Данная методика была внедрена в один из проектов компании ООО «НетКрэкер». На примере одной из команд удалось проанализировать эффективность данной методики, в результате чего стало видно, что, во-первых, на этапе интеграционного тестирования тестировщики стали находить меньшее количество ошибок, чем находили до внедрения данной методики в проект, это произошло благодаря тому, что команда разработки стала самостоятельно проводить модульное тестирование и «отлавливать» ошибки до того, как функционал попадет к тестировщикам. Помимо этого, на графиках видно, что при работе по данной методике, уменьшилась нагрузка на команду тестирования, а также уменьшилось количество переоткрываемых ошибок.

В то же время данная методика не исключает ряд недостатков, например, с использованием данной методики растет нагрузка на команду разработчиков, из-за чего снижается количество выдаваемого функционала.

Помимо этого, был представлен шаг по улучшению данной методики – переход команды разработки на интеграционное тестирование, что, возможно, еще больше повысит качество поставляемого функционала

Заключение

В работе был сделан вывод, что, в настоящее время не существует единого определения для такого термина, как «методика тестирования», в результате чего было предоставлено собственное определение данному термину – это конкретные действия, шаги для достижения контроля качества программного обеспечения.

Был сделан вывод о том, что некоторые из них должны быть покрыты новой методикой тестирования, например, интеграционное и системное тестирование, а также тестирование документации.

В результате, проанализировав существующие методы и виды тестирования, в работе была представлена и опробована на реальной системе новая методика тестирования, содержащая следующие шаги:

- *сбор требований о тестируемой системе* во время которого была предоставлена реальная система для тестирования, а также собраны требования для нового функционала;
- *анализ требований*, которые были проанализированы с точки зрения адекватности, полноты, совместимости, выполнимости и разумности, а также с точки зрения подверженности тестированию;
- шаг *разработки тестовых сценариев* включал составление примеров тестовых сценариев для стартовой страницы одного из зарубежных телекоммуникационных операторов с помощью методологии случаев использования, техники классов эквивалентности и анализа пограничных значений, а также анализ покрытия кода;
- интеграционное *тестирование* и последующее *отслеживание найденных ошибок* – представлял собой тестирование взаимодействия двух систем. Были подробно протестированы составленные ранее тестовые сценарии, в результате чего была найдена и зафиксирована ошибка;
- *тестирование пользовательского интерфейса*, в ходе которого были протестированы различные характеристики графического интерфейса, после чего был сделан вывод, что он соответствует требованиям и

эстетическим желаниям пользователей, даже несмотря на то, что был найден один незначительный недостаток;

- шаг *системное тестирование* включал проверку различных позитивных и негативных end-to-end процессов;
- *регрессионное тестирование* включало в себя последние прогоны основных бизнес-сценариев, после чего новый функционал был готов к выдаче;
- *выдача* проекта – шаг, на котором были написаны рекомендации по установке сборки и по настройке нового функционала на реальном сервере.

Применив приведенную методику на реальной системе, а также на реальном проекте разработки ПО можно сделать вывод, что данная методика:

1) может применяться для тестирования различных проектов по разработке ПО, однако более подходит для проектов, работающих по гибким методологиям управления;
2) дает конкретные шаги тестирования, поэтому может помочь начинающим тестировщикам легче «влиться» в процесс;
3) может применяться и для опытных команд тестировщиков, как это было сделано к одной из команд компании «НетКрэкер».

Таким образом, нам удалось проанализировать эффективность данной методики, в результате чего стало видно, что:

1) улучшилось *качество* разрабатываемого *функционала* значительно за счет нахождения меньшего количества ошибок во время интеграционного тестирования, а также таких факторов как процент переоткрытых ошибок, непройденных тестовых сценариев после первого прогона тестов;
2) снизилась нагрузка на команду тестировщиков за счет проведения модульного тестирования разработчиками.

Данная методика не исключает ряд недостатков: еще большее улучшение качества поставляемого функционала посредством частичного или полного перевода команды разработчиков на шаг интеграционного тестирования.

Список используемой литературы

Нормативно-правовые акты:

1. ГОСТ Р 50922-2006. Защита информации. Основные термины и определения
2. ГОСТ Р ИСО/МЭК 12207-2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств
3. Методология функционального моделирования IDEF0, Руководящий документ, Госстандарт России

Научная и методическая литература:

4. Абезгауз Д.Б. Редактор диаграмм функциональных блоков программируемых контроллеров // Вестник Пензенского государственного университета. 2013. №3. С. 66-69.
5. Антонова Г.М. Эволюция терминов «Черный ящик» и «Серый ящик» // Вестник Московского финансово-юридического университета. 2012. №1. С. 16-19.
6. Аткинсон Л. MySQL. Библиотека профессионала - СПб: Вильямс, 2014. - 624 с.
7. Барахтаев М.А., Демкин К.В., Харитонов А.Ю. Проблемы тестирования программного обеспечения встроенных систем // Международный журнал гуманитарных и естественных наук. 2018. №5-1. С. 184-187.
8. Бедердинова О.И., Иванова Л.А. Совершенствование метода тестирования программного обеспечения «Белый ящик» // Arctic Evironmental Research. 2014. №2. С. 113-123.
9. Бирюков С.В. Анализ стратегий тестирования программного обеспечения // Известия Южного федерального университета. Технические науки. 2008. №1 (78). С. 59-63.
10. Бритов Г., Осипова Т. Моделирование бизнес-процессов. - М.:LAP, 2014. – 124 с.

11. Варфоломеева Е.В. Информационные системы в экономике: Учебное пособие / Е.В. Варфоломеева, Т.В. Воропаева и др.; Под ред. Д.В. Чистова - М.: НИЦ ИНФРА-М, 2015. - 234 с.
12. Вдовенко Л.А. Информационная система предприятия: Учебное пособие/Вдовенко Л. А. - 2 изд., перераб. и доп. - М.: Вузовский учебник, НИЦ ИНФРА-М, 2015. - 304 с.
13. Гвоздева В.А. Базовые и прикладные информационные технологии: Учебник / Гвоздева В. А. - М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2015. - 384 с.
14. Гвоздева В.А. Информатика, автоматизированные информационные технологии и системы: Учебник / В.А. Гвоздева. - М.: ИД ФОРУМ: НИЦ ИНФРА-М, 2015. - 544 с.
15. Гладышев Е.И., Мурыгин А.В. Обзор архитектура современного OSS/BSS решения // Актуальные проблемы авиации и космонавтики. 2011. №7(1). С. 386-387.
16. Грофф Д., Вайнберг П., Оппель Э. SQL. Полное руководство. - СПб.: Вильямс, 2014. - 960с.
17. Дейт К.Дж. Введение в системы баз данных. - К.: Диалектика, 2012. - 360 с.
18. Домарацкий Я.А. Модульное тестирование операционной системы реального времени // Программные продукты и системы. 2017. №4. С. 37-40.
19. Дунаев В.В. Базы данных. Язык SQL для студента – Издательство: БХВ, 2013. - 196 с.
20. Журавлев. С.О., Никитина С.А. Скорынин А.С. Об одном подходе к разбиению на классы эквивалентности // Челябинский физико-математический журнал. 2017. №4. С. 497-502.
21. Заботина Н.Н. Проектирование информационных систем: Учебное пособие / Н.Н. Заботина. - М.: ИНФРА-М, 2011. - 331 с.
22. Затонский А.В. Информационные технологии: разработка информационных моделей и систем: Учеб. пос. / А.В.Затонский - М.: ИЦ РИОР: НИЦ ИНФРА-М, 2014 - 344с.

23. Информационные системы в экономике: Учебник / К.В. Балдин, В.Б. Уткин. - 7-е изд. - М.: Дашков и К, 2012. - 395 с.
24. Исаев Г. Проектирование информационных систем : учебное пособие : М.: Омега-Л, 2015. 432с.
25. Калиберда Е.А., Федотова И.В. Анализ требований к программным продуктам с выбором метода тестирования на примере web-ориентированных приложений // Омский научный вестник. 2017. №2 (110). С. 259-262.
26. Климентьев К.Е. Современные методы и технологии тестирования программного обеспечения: электрон. учеб.-метод. комплекс по дисциплине в LMS Moodle : мин-во образования и науки РФ, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т), 2013. 43 с.
27. Коваленко В.В. Проектирование информационных систем: Учебное пособие / В.В. Коваленко. - М.: Форум: НИЦ ИНФРА-М, 2014. - 320 с.
28. Костомаров В.Г., Митрофанова О.Д. Методика как наука. Статья 1 // ФГБОУ ВО «Государственный институт русского языка им. А.С. Пушкина». 2017. №3. С 56-62.
29. Котляров В.П. Критерии покрытия требований в тестовых сценариях, сгенерированных из поведенческих моделей приложений // Научно-технические ведомости Санкт-Петербургского государственного политехнического университета. Информатика, телекоммуникации и управление. 2016. №6(1). С 202-207.
30. Куликов С. Тестирование программного обеспечения. Базовый курс: учеб. Пособие : EPAM Systems, 2020. 298 с.
31. Мутилин В.С. Паттерны проектирования тестовых сценариев // Труды Института системного программирования РАН. 2018. №5. С 97-128.
32. Никифоров И.В., Петров А.В., Котляров В.П. Статический метод отладки тестовых сценариев, сгенерированных с использованием эвристик // Научно-технические ведомости Санкт-Петербургского государственного

политехнического университета. Информатика, телекоммуникации и управление. 2012. №4. С 114-119.

33. Носырев П., Носырева М., Рассказова Т., Корнеева Н. Валидация аналитических методик: теория и практика (часть I. теория) // Ремедиум. Журнал о российском рынке лекарств и медицинской технике. 2016. №11. С 62-64.
34. Петров А.С., Плаксин М.А., Сергеев Д.И. Автоматизация контроля обучения студентов тестированию учебных программ методами «чёрного ящика» // Вестник Пермского университета. Серия: Математика. Механика. Информатика. 2010. №2. С. 85-93.
35. Пирогов, В.Ю. Информационные системы и базы данных: организация и проектирование: Учебное пособие / В.Ю. Пирогов. - СПб.: БХВ-Петербург, 2014. - 528 с.
36. Полевщиков И.С. Особенности разработки методического пособия на тему «Разбиение по эквивалентности и анализ граничных значений» (для студентов направлений «Информатика и вычислительная техника» и «Программная инженерия») // Инновации в науке. 2013. №18-1. С. 1-6.
37. Полухин П.В. Практическая апробация фаззинга межсайтового скриптинга методом черного ящика к тестированию интернет и интранет приложений // Перспективы развития информационных технологий. 2015. №25. С. 17-21.
38. Поначугин А.В. Определение надёжности программного обеспечения в структуре современной информационной системы // Кибернетика и программирование. 2016. №2. С. 65-72.
39. Уткин В., Балдин К. Информационные системы в экономике. - М.: Academia, 2014. - 288с.
40. Федотова Е.Л. Информационные технологии в профессиональной деятельности: Учебное пособие / Е.Л. Федотова. - М.: ИД ФОРУМ: НИЦ ИНФРА-М, 2015. - 368 с.

41. Фуфаев, Э.В. Базы данных: Учебное пособие для студентов учреждений среднего профессионального образования / Э.В. Фуфаев, Д.Э. Фуфаев. - М.: ИЦ Академия, 2014. - 320 с.

Электронные ресурсы:

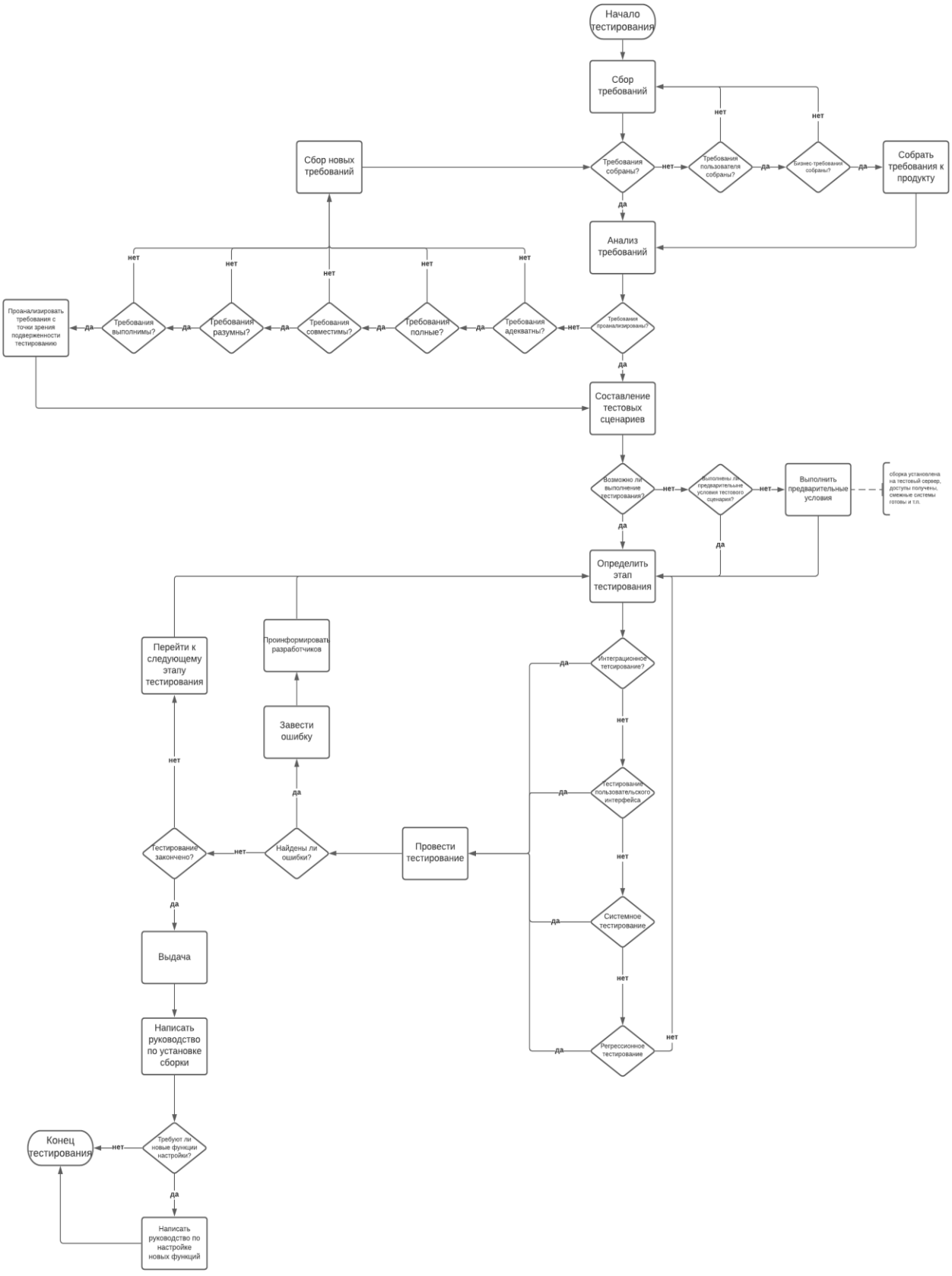
42. Фуфаев, Э.В. Базы данных: Учебное пособие для студентов учреждений среднего профессионального образования / Э.В. Фуфаев, Д.Э. Фуфаев. - М.: ИЦ Академия, 2014. - 320 с.

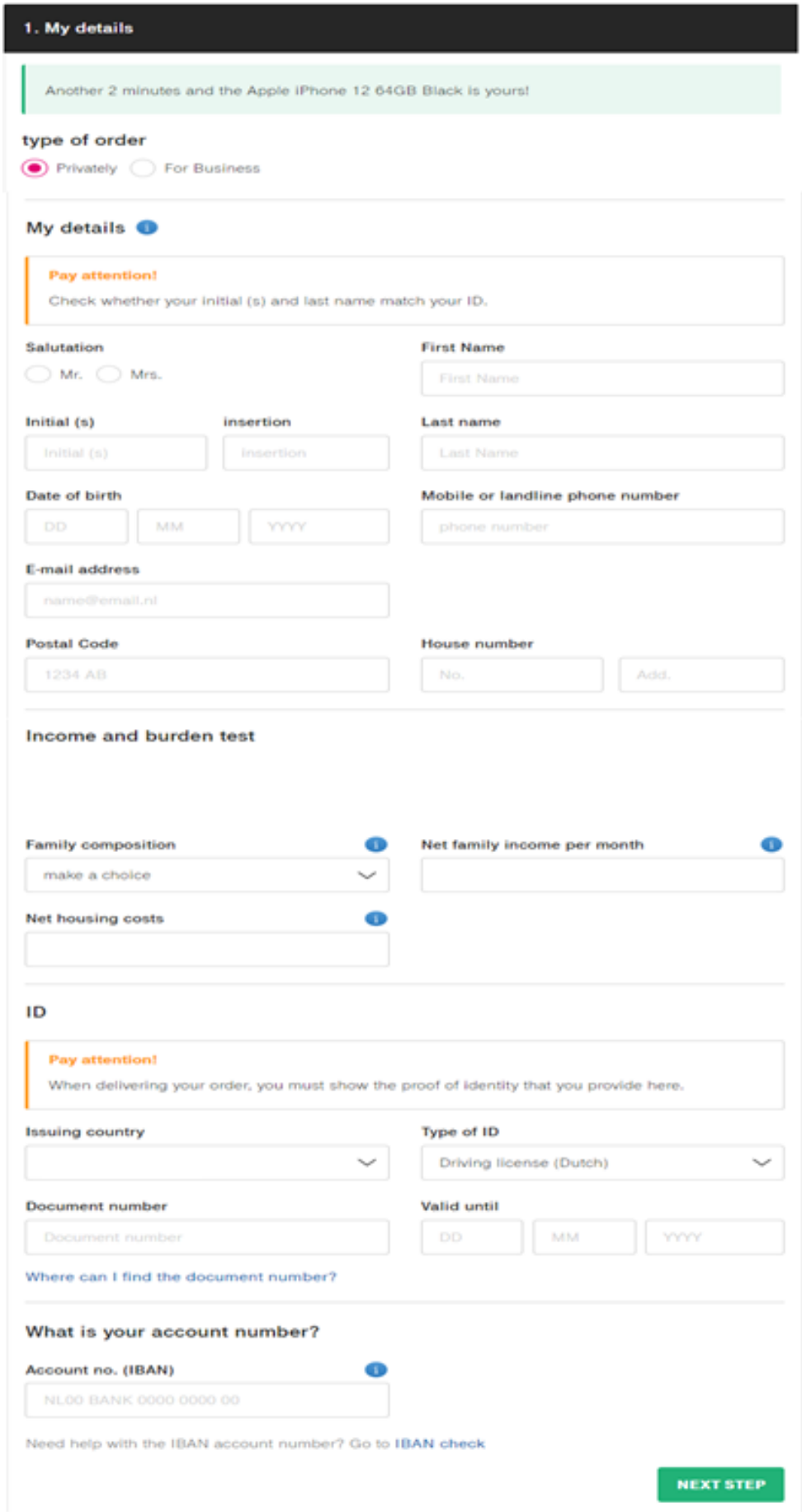
Литература на иностранном языке:

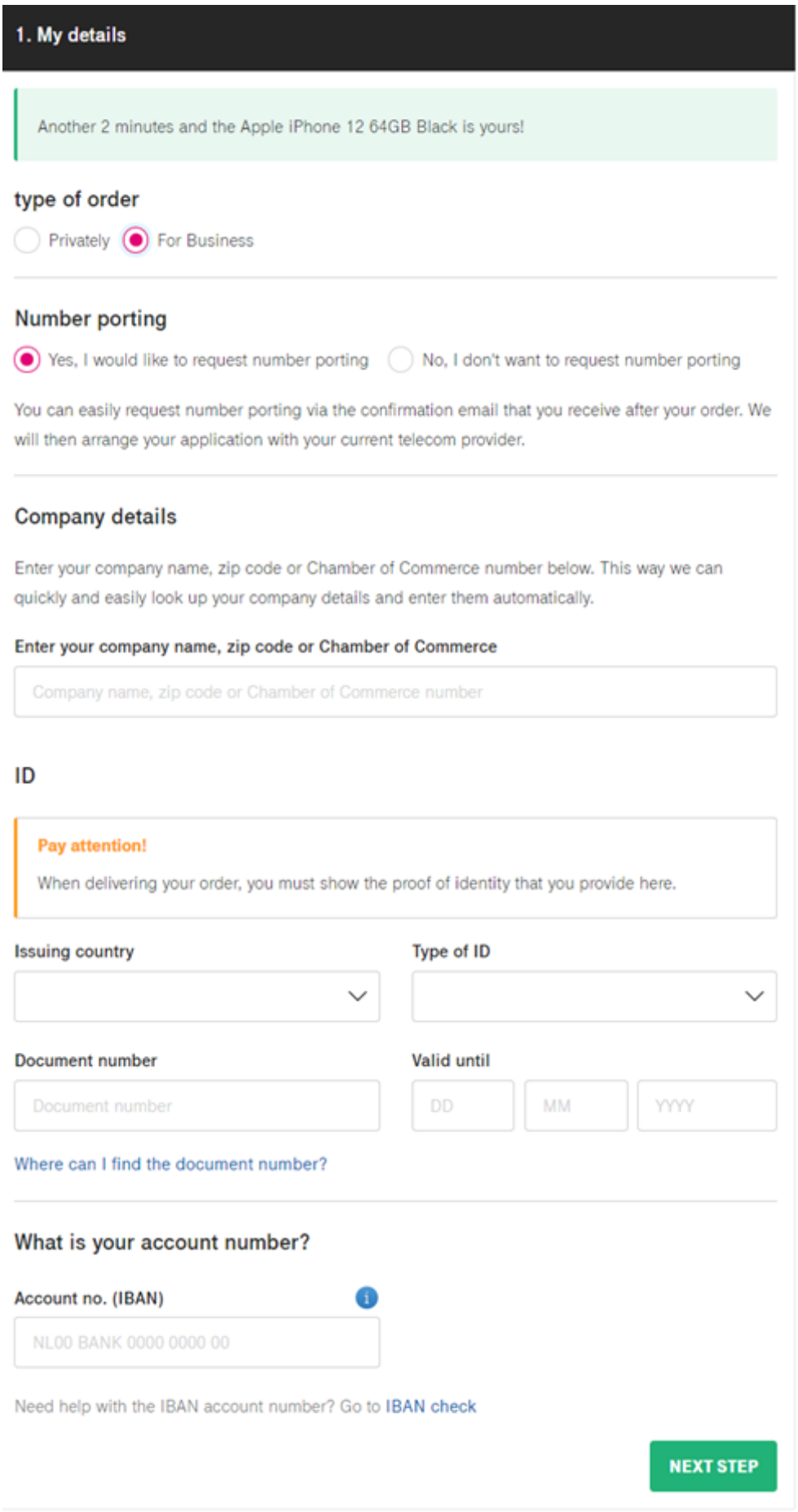
43. Copeland L. A practitioner’s Guide to Software Test Design : Artech House Publishers, 2003. 300 p.
44. Kaner C., Falk J., Nguyen H.Q. Testing computer software : Wiley, 1999. 496.
45. Luciano B., Mauro P. Electronic Notes in Theoretical Computer Science // ELSEVIER. 2015. №148. P.89-111.
46. Reddy N. The Difference In Perspective Of “Testers” And “Developers” // Software Testing Help. 2020. №3. P 18-22.
47. Xu Q., Jiao R.J., Yang X., Helander M.G. Customer Requirement Analysis based on an Analytical Kano Model // School of Mechanical & Aerospace Engineering, Nanyang Technological University, Singapore. 2008. №1. P 7-12.

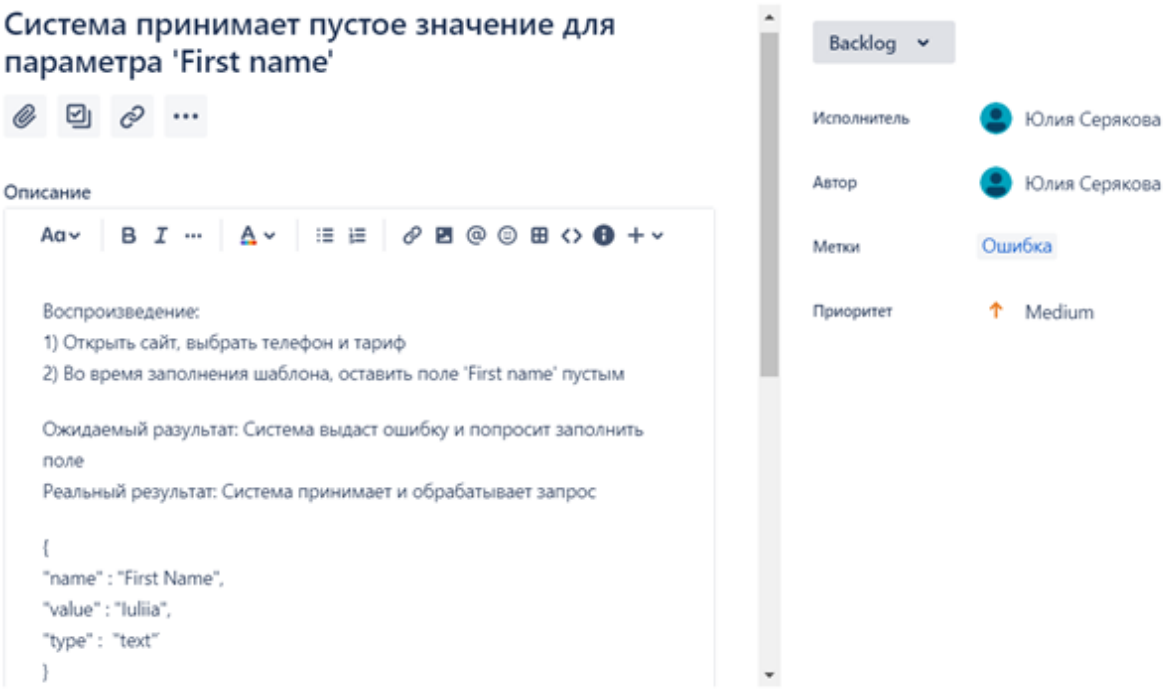
Приложение А

Схемы и экранные формы




1. My details
Another 2 minutes and the Apple iPhone 12 64GB Black is yours!
type of order
Privately
For Business
My details
Pay attention!
Check whether your initial (s) and last name match your ID.
Salutation
Mr.
Mrs.
First Name
First Name
Initial (s)
insertion
Last name
Initial (s)
insertion
Last Name
Date of birth
DD
MM
YYYY
Mobile or landline phone number
phone number
E-mail address
name@email.nl
Postal Code
1234 AB
House number
No.
Add.
Income and burden test
Family composition
make a choice
Net family income per month
Net housing costs
ID
Pay attention!
When delivering your order, you must show the proof of identity that you provide here.
Issuing country
Type of ID
Driving license (Dutch)
Document number
Document number
Valid until
DD
MM
YYYY
Where can I find the document number?
What is your account number?
Account no. (IBAN)
NL00 BANK 0000 0000 00
Need help with the IBAN account number? Go to IBAN check
NEXT STEP


1. My details
Another 2 minutes and the Apple iPhone 12 64GB Black is yours!
type of order
Privately
For Business
Number porting
Yes, I would like to request number porting
No, I don't want to request number porting
You can easily request number porting via the confirmation email that you receive after your order. We will then arrange your application with your current telecom provider.
Company details
Enter your company name, zip code or Chamber of Commerce number below. This way we can quickly and easily look up your company details and enter them automatically.
Enter your company name, zip code or Chamber of Commerce
Company name, zip code or Chamber of Commerce number
ID
Pay attention!
When delivering your order, you must show the proof of identity that you provide here.
Issuing country
Type of ID
Document number
Valid until
Document number
DD
MM
YYYY
Where can I find the document number?
What is your account number?
Account no. (IBAN)
NL00 BANK 0000 0000 00
Need help with the IBAN account number? Go to IBAN check
NEXT STEP


Система принимает пустое значение для параметра 'First name'
Описание
Aa
Воспроизведение:
1) Открыть сайт, выбрать телефон и тариф
2) Во время заполнения шаблона, оставить поле 'First name' пустым
Ожидаемый разультат: Система выдаст ошибку и попросит заполнить поле
Реальный результат: Система принимает и обрабатывает запрос
{
"name" : "First Name",
"value" : "Iuliia",
"type" : "text"
}
Backlog
Исполнитель
Юлия Серякова
Автор
Юлия Серякова
Метки
Ошибка
Приоритет
Medium