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)

Технология бизнес-анализа
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему Методы и инструменты многокритериального оценивания качества релизов программного обеспечения

Обучающийся Е. А. Филимонова
(Инициалы Фамилия) (личная подпись)

Научный руководитель доктор социологических наук, доцент, Е. В. Желнина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2023

Оглавление

Введение

Возникновение темы тестирования программного обеспечения пришлось на конец XX века. Бурное развитие различных технологий привело к росту рынка производства программного обеспечения и к пересмотру вопросов обеспечения качества и надёжности разрабатываемых программных обеспечений.

Актуальность исследования магистерской диссертации определяется несогласованностью общепринятых теоретических концепций о классической подготовке технической документации проекта по разработке программного обеспечения, а также организации и реализации этапа тестирования как оценки качества планируемого продукта и применения методик устранения найденных дефектов продукта.

Научная проблема исследования заключается в том, что современному специалисту предоставлена широкая база классических сочинений по теории о процессе тестирования программного обеспечения на основе спецификаций и дизайнов, сформированных по собранным требованиям заказчика к продукту, однако возникают значительные трудности с поиском практического решения возникающих проблем, связанных непосредственно с процессом обращения к документации на каждом этапе разработки проекта.

Цель исследования в рамках данной магистерской диссертации состоит в изучении практических методов и инструментов многокритериального оценивания качества релизов программного обеспечения и в их применении в разработке и тестировании приложения «Menu 1.0», необходимого для автоматизации деятельности организации общественного питания.

Объектом исследования в данной работе выступает процесс планирования и реализации тестирования релиза программного обеспечения.

Предметом исследования являются методы и инструменты многокритериального оценивания качества релиза программного обеспечения, используемые в процессе тестирования.

Гипотеза исследования состоит в предположении, что применение выявленных методов и инструментов в рамках данной магистерской диссертации позволит облегчить процесс тестирования программного обеспечения и улучшить качество выдаваемого релиза программного обеспечения, в том числе обеспечить разработку приложения «Menu 1.0» для автоматизации деятельности организации общественного питания.

Для достижения поставленной цели данной магистерской диссертации необходимо решить следующие задачи:

– проведение анализа литературы и источников, соответствующих выбранной теме исследования;

– формирование теоретических знаний о процессе тестирования как этапа методологии разработки программного обеспечения;

– рассмотрение основных концепций оценки качества релиза;

– изучение классификации методов тестирования программного обеспечения и их особенностей;

– формирование знаний о технической документации проекта разработки программного обеспечения;

– проведение анализа основных методик разработки плана и стратегии тестирования программного обеспечения;

– изучение классификации дефектов программного обеспечения, найденных в процессе тестирования;

– выявление эффективных методов и способов устранения дефектов релиза программного обеспечения;

– рассмотрение возможных последствий и влияния на программное обеспечение, возникших при упущении дефектов на различных этапах разработки;

– применение выявленных методов и инструментов на примере реализации оценки качества релиза приложения для предприятия общественного питания.

Наиболее значимой теоретической основой данного исследования выступают работы и труды российских и зарубежных специалистов в области изучения процесса оценки и обеспечения качества релиза программного продукта на основе спецификаций и дизайнов. Опыт, полученный на производственной практике, поможет сформировать практическую структуру профессиональных знаний и возможных действий, которые в настоящее время не опубликованы специалистами, а также привести реалистичный пример процесса обеспечения качества релиза приложения путем регулярного обращения к данным документам.

Методами исследования являются сравнительный анализ, методы тестирования программного обеспечения, эксперимент.

Апробация результатов исследования представлена в статье: Филимонова Е.А. Особенности разработки плана тестирования программного обеспечения с целью максимального покрытия решений / Е.А. Филимонова // Молодежь. Наука. Общество – 2021.

Исследование проводилось с 2021 по 2023 годы в несколько этапов:

На первом этапе формулировалась тема исследования и выполнялся сбор и анализ теоретических данных по теме исследования, также проводилась постановка цели, задач, предмета, объекта и гипотезы исследования.

На втором этапе осуществлялся анализ стадий формирования технической документации программного обеспечения, методов тестирования программного обеспечения и классификации выявленных дефектов, создавался алгоритм обеспечения качества релиза, публиковался результат исследования.

На третьем этапе предлагаемые методы и инструменты многокритериального оценивания качества релиза применялись в реализации приложения «Menu 1.0», были сформулированы выводы о полученных результатах по проведенному исследованию.

Научная новизна работы заключается в научном обосновании методов и инструментов многокритериального оценивания качества релизов

программного обеспечения для планирования и реализации этапа тестирования программного обеспечения на примере разработки приложения «Menu 1.0» для автоматизации деятельности организации общественного питания.

Теоретическая значимость работы заключается в том, что в ней проанализированы основные существующие классификации методов тестирования и обеспечения качества релиза программного обеспечения.

Практическая значимость результатов данной магистерской диссертации заключается в сокращении трудоемкости и сроков подготовки и реализации этапа тестирования программного обеспечения, а также в рациональном покрытии диапазона решений программного обеспечения за счет предлагаемых методов и инструментов многокритериального оценивания качества релиза на примере приложения «Menu 1.0».

Положения, выносимые на защиту:

– благодаря системе отслеживания ошибок, которая способствует своевременному реагированию на найденные ошибки и их быстрому устранению, упрощается и систематизируется процесс тестирования и обеспечения качества программного обеспечения;

– этапы разработки приложения «Menu 1.0» для автоматизации деятельности организации общественного питания по каскадной методологии разработки проекта;

– моделирование бизнес-процесса организации общественного питания в рамках методологии BPMN «как есть» и «как будет» при внедрении приложения «Menu 1.0» для автоматизации деятельности;

– декомпозиция и анализ бизнес-процесса приложения «Menu 1.0» для автоматизации деятельности организации общественного питания;

– разработанные план и стратегия тестирования приложения «Menu 1.0» для автоматизации деятельности организации общественного питания;

– применение методов функционального тестирования, позитивного тестирования, негативного тестирования, нагрузочного тестирования, стресс-

тестирования и тестирования удобства пользователя для реализации приложения «Menu 1.0»;

– результаты применения методов и инструментов многокритериального оценивания качества релиза приложения «Menu 1.0» в виде статистических отчетов и графиков;

– дефекты приложения «Menu 1.0» выявлены в результате прохождения тестовых сценариев на основе позитивного и негативного методов тестирования, а также метода тестирования удобства пользователя.

Магистерская диссертация состоит из введения, трех глав и заключения, изложенных на 90 страниц, а также 18 рисунков, 14 таблиц и списка использованной литературы, состоящего из 32 наименований, и 2 приложений.

Глава 1 Теоретические аспекты тестирования и обеспечения качества программного продукта

1.1 Значение качества разработки программного обеспечения в каскадной методологии

В настоящее время существует множество методологий разработки программного обеспечения, в каждой из которых выделяется обязательный этап тестирования. Также на протяжении всего цикла разработки проекта принимает участие команда по обеспечению качества каждого из этапов.

Методологии реализуются на основе итеративного или инкрементного подходов. Главным отличием данных подходов является особенность процесса реализации: по итеративному подходу проект реализуется поэтапно с равномерным улучшением и детализацией, а инкрементный подход подразумевает разработку каждой части программного обеспечения сразу до ожидаемого вида [4]. Наглядный пример отличия итеративной и инкрементной моделей представлен на рисунке 1. Стоит учесть, что и итеративная, и инкрементная модели в конечном счете будут иметь одинаковый результат.

К традиционным итеративным подходам относятся такие методологии, как каскадная модель, инкрементная модель, спиральная модель и V-образная модель. К подтипам гибких инкрементных методологий относятся такие Agile[1] подходы, например, как Scrum, Kanban, FDD, DSDM, XP, Lean, ASD и Crystal. Основная цель данных методологий заключается в обеспечении бесперебойной разработки проекта в соответствии с требованиями заказчика [21].

[1] Agile – это гибкий подход к управлению проектами и разработке программного обеспечения, направленный на важность взаимодействия людей и постоянное сотрудничество с заказчиком.

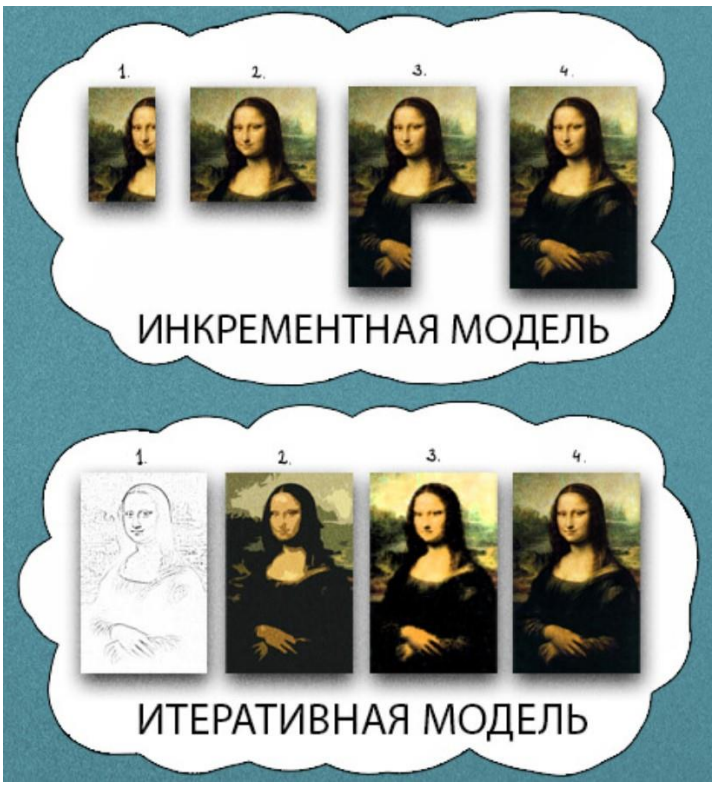


Рисунок 1 – Отличие итеративной и инкрементной моделей разработки программного обеспечения

Методология разработки программного обеспечения – это основа, которая используется для структурирования, планирования и управления процессом разработки проекта. Методологии касаются только самого процесса разработки программного обеспечения, поэтому они не включают никаких технических аспектов, а касаются исключительно правильного планирования разработки.

В программной инженерии процесс разработки программного обеспечения – это процесс разделения работы на отдельные фазы, который также известен как жизненный цикл разработки программного обеспечения. Методология может включать предварительное определение конкретных результатов и артефактов, которые создаются и завершаются командой проекта для разработки или сопровождения программы.

В данной выпускной квалификационной работе рассмотрим каскадную модель разработки для дальнейшего применения на примере проекта приложения «Menu 1.0».

Каскадная модель, также называемая Waterfall (Водопад), является одной из самых первых методологий разработки проекта. На основе сравнения с потоком воды, данная модель направляет команду к решению задач по намеченному в начале плану и строго последовательно. При использовании данной методологии легко управлять проектом. Благодаря своей жесткости разработка происходит быстро, стоимость и время разработки предопределены [11].

Современная методология Waterfall является более гибкой, однако также подразделяет процесс разработки проекта на строгие основные этапы, изображенные визуально в виде схемы на рисунке 2 [31].

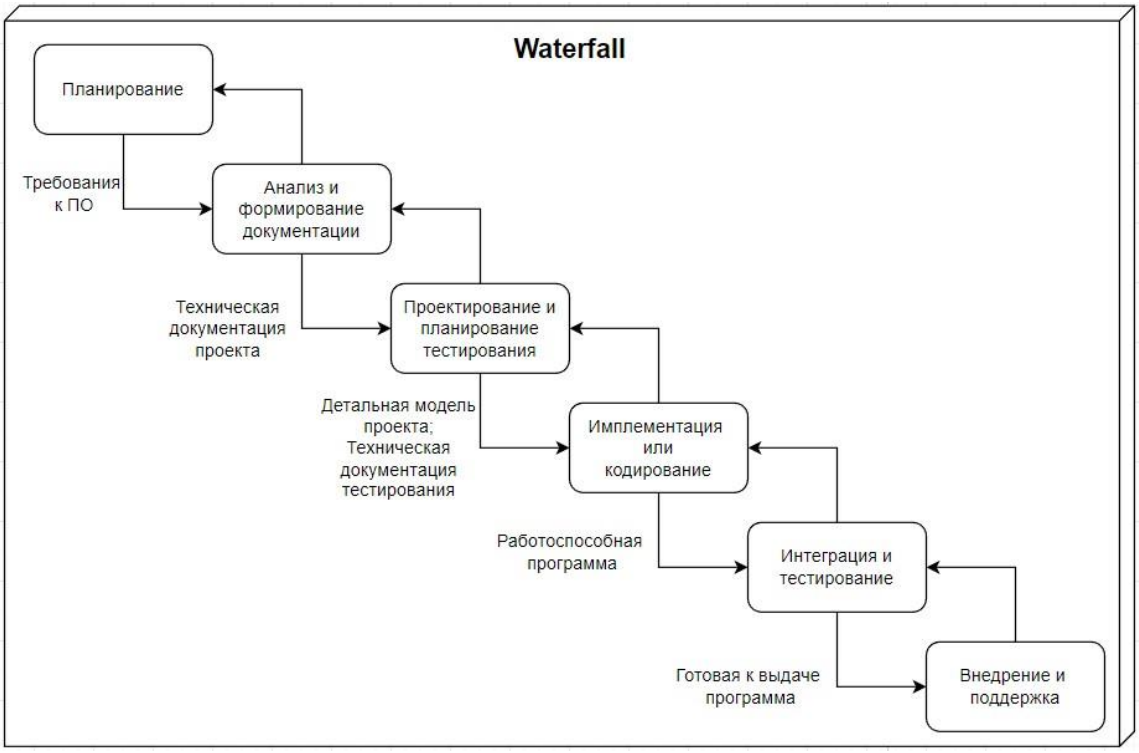


Рисунок 2 – Каскадная модель разработки программного обеспечения

Разберем подробнее каждый этап каскадной модели.

На этапе планирования команда бизнес-аналитиков собирает пожелания и требования заказчика, по которым создается входная документация как основа для дальнейшего формирования технической документации проекта. На данном этапе уже подключаются руководители команды разработки и обеспечения качества. Результатом данного этапа выступает готовый перечень требований к ожидаемому программному обеспечению.

Этап анализа и формирования документации подразумевает, что команда бизнес-аналитиков, опираясь на собранную информацию в предыдущей фазе, приступает к написанию документации. Каскадная модель предполагает тщательное и детальное описание каждого элемента программного обеспечения, а также выбор конкретных инструментов для разработки программного обеспечения и языка программирования. С точки зрения обеспечения качества проектной документации к данному этапу подключается и руководитель команды тестирования, и аналитик обеспечения качества. Результатом данного этапа является составленная техническая документация проекта.

На этапе проектирования и планирования тестирования реализуется параллельно две деятельности команды. Бизнес-аналитики и руководители проекта приступают к моделированию будущего программного обеспечения, представляя в виде эскизов, макетов, схем для формирования визуальной общей картины. Команда по обеспечению качества и ее руководитель формируют документацию для дальнейшего тестирования работоспособного программного обеспечения, разрабатывают все возможные сценарии тестирования решения. По завершению данного этапа основная часть бизнес-аналитиков покидает проект и подключается команда разработки решения.

Этап имплементации решения выполняется командой разработки, которая реализует программное обеспечение на основе технической документации. Результатом данного этапа выступает работоспособная программа, допущенная к следующему этапу, при этом основная часть

команды разработки покидает проект и остаются только некоторые участники разработки для дальнейшего устранения найденных ошибок.

На этапе интеграции выполняется подготовка тестовой среды, объединение разработанных модулей программы и далее запускается процесс тестирования программного обеспечения. На текущем этапе привлекается вся команда по обеспечению качества и проводится проверка каждого разработанного тестового сценария. Специалисты по обеспечению качества анализируют результаты и осуществляют поиск дефектов программы, опираясь на созданную в предыдущих этапах документацию. В случае обнаружения команда тестирования передает информацию об ошибке команде разработки. Данный этап позволяет осуществить необходимый возврат проекта к предыдущему этапу для устранения дефектов. Результатом данного этапа является готовое программное обеспечение без критических и блокирующих проблем, наличие которых не позволяет переходить к следующему этапу каскадной модели. Именно этап тестирования играет ключевую роль в разработке проекта, потому что только на данном этапе проводится оценка фактического состояния качества релиза.

Этап внедрения и поддержки начинается, когда готовое решение передается заказчику для установки на внешнюю среду и эксплуатации пользователями. На протяжении всего этапа команда поддержки качества принимает участие и несет ответственность за результат. В случае выявления дефектов участники проекта могут возвращаться к предыдущим этапам для устранения проблем и поддержки стабильного состояния программного обеспечения, а также разработки небольших изменений.

Большинство проектов, использующих каскадную модель разработки, в настоящее время включает все вышеперечисленные этапы классической модели, при этом используя возможность работать разным командам над разными этапами одновременно [8].

Первоначальные принципы каскадной методологии были более строгие и ограничивали работу команды и требования самого заказчика:

– каждое условие и требование к системе должно быть зафиксировано в технической документации проекта;

– каждый последующий этап начинается только по завершении предыдущего этапа;

– пропуск любого из этапов разработки проекта невозможен;

– возвращение к предыдущему этапу для внесения изменений невозможно;

– при внесении изменений в требования к системе после согласования заказчиком необходимо пересматривать и переписывать техническую документацию;

– выявление и исправление дефектов системы возможно только на этапе тестирования программного обеспечения;

– заказчик проекта не принимает участия в процессе разработки программного обеспечения после завершения этапа согласования технической документации.

Методология разработки Waterfall является негибкой моделью, однако обладает преимуществами для определённых проектов [8]. К ним можно отнести следующие факты:

– каскадная модель подходит для участия в разработке проекта с большими командами;

– так как есть готовая техническая документация, то можно разбить проект на отдельные этапы разработки. В каскадной модели есть возможность распределить задачи каждого этапа на разные команды и при помощи качественно продуманной документации каждая команда будет знать какие конкретно модули с какими характеристиками нужно создать и в какой срок;

– каскадная модель устойчива к изменениям кадрового состава. Любой участник процесса разработки может меняться и не повлияет на качество и сроки выполнения задач благодаря подробно подготовленной документации;

– каскадная модель позволит планировать процесс разработки на длительный срок, так как техническая документация проекта уже готова, заказчик не собирается участвовать далее в процессе реализации проекта и, соответственно, значительные изменения документации не ожидаются;

– на основе каскадной модели проект реализуется последовательно, а значит по завершении каждой стадии разработки заказчик сможет видеть результат проделанной работы.

При выборе каскадной модели для разработки программного обеспечения выделяются следующие недостатки и ограничения модели:

– если в проекте ожидается большой объем функциональной части, то объем сопровождающей необходимой документации будет также значительно огромным;

– каскадная модель дает отличный результат только в проектах с четко определенными требованиями и методами их реализации;

– при использовании каскадной модели не допускается какие-либо отступления от намеченного результата в процессе разработки;

– любые изменения приведут к дорогостоящим последствиям, так как необходимо дожидаться завершения всех этапов разработки для инициализации запроса на внесение изменений;

– в соответствии с каскадной моделью этап тестирования ожидается только после завершения или почти завершения стадии конструирования. Поэтому дефекты и несоответствия требованиям могут быть найдены слишком поздно;

– Заказчик недостаточно заинтересован промежуточными стадиями разработки, поэтому проект в результате может получится отличным от первоначальных ожиданий заказчика. Из-за незаинтересованности заказчика, возможно, не будет налажена обратная связь.

Итак, любая методология подбирается и применяется для определенного вида проекта и не существует универсальной модели разработки

программного обеспечения. Каждая методология уникальна своими преимуществами и недостатками, однако процесс обеспечения качества проекта прослеживается в каждой методологии и направляет проект к изначально намеченному результату [20].

Таким образом, обеспечение качества каждого этапа играет важную роль в разработке программного обеспечения вне зависимости от выбранной методологии. Для разработки любого проекта важно правильно выбрать модель реализации для эффективной организации участников проекта и грамотного управления этапами разработки. Оценка качества должна осуществляться с ранних фаз проекта с целью превентирования дальнейших ошибок и проблем программного обеспечения, а также для удовлетворения требований заказчика проекта. Тестирование как этап разработки в каскадной модели является ключевым, потому как именно команда по обеспечению качества может сделать вывод о соответствии программного обеспечения требованиям заказчика и выдвинуть решение о допуске программного обеспечения к эксплуатации.

1.2 Основные концепции оценки качества программного обеспечения

В настоящее время в процессе реализации запланированного проекта современные IT-компании[2] выделяют тестирование как неотъемлемую часть на каждом этапе жизненного цикла разработки программного обеспечения. Процесс тестирования позволяет избежать негативных последствий как для проекта, так и для бизнеса, а также обеспечить соответствующее качество, исключая человеческий фактор в лице команды разработчиков.

Тестирование представляет собой процесс на основе исследовательского метода, позволяющий проводить испытание

[2] IT-компания – это компания, реализующая деятельность в сфере современных информационных технологий.

программного обеспечения с целью реализации проверки соответствия подготовленных требований проекта с фактическим состоянием разработанного программного обеспечения, а также устранения найденных несоответствий.

Следует учитывать, что процесс обеспечения качества и процесс тестирования отличаются базовыми концепциями. Концепция тестирования заключается в улучшении качества программного обеспечения путем обнаружения дефектов фактически разработанного продукта, а концепция обеспечения качества направлена на улучшение процесса разработки продукта с первого этапа жизненного цикла с целью предотвращения возможных дефектов на этапе тестирования. [6]

Основной целью обеспечения качества продукта в процессе тестирования является поиск дефектов на основе следующих критериев:

– специалист по обеспечению качества получает ожидаемый результат от воспроизведения необходимого сценария[3] в тестируемой программе;

– специалист по обеспечению качества получает фактический результат от воспроизведения необходимого сценария в тестируемой программе;

– специалист по обеспечению качества проводит сравнение первого и второго критерия.

К основным источникам ожидаемого результата тестирования программного обеспечения относятся техническая документация (дизайн, спецификация), статистические данные, стандарты IT-компании, законодательство, жизненный опыт каждого участника реализации проекта и, безусловно, здравый смысл.

В состав технической документации входит документ, содержащий подробное описание работы каждой детали планируемого программного

[3] Сценарий в тестировании – это фиксированная последовательность действий для получения фактического результата проверки программного обеспечения.

обеспечения, называемый спецификацией. В процессе подготовки к этапу тестирования программы команда по обеспечению качества разрабатывает стратегию и план тестирования на основе вышеуказанной спецификации, а в процессе реализации тестирования также регулярно обращается к спецификации. [2]

Документ, описывающий внешний вид пользовательского интерфейса, включая приоритетные цвета, размеры, структуру модулей и последовательность действий пользователя, формируется на основе требований заказчика проекта и устава его организации как дизайн пользовательского интерфейса.

Рассмотрим пример постановки одного из пунктов дизайна интерфейса на основе бизнес-требования заказчика. Логотип организации заказчика выполнен в оранжевом, красном и белом цвете, соответственно проект, разработанный для автоматизации бизнес-процесса, должен быть выполнен в тех же цветах, а именно цвет кнопок, строк, фона программы.

На этапе формирования технической документации проекта, которая играет основную роль для дальнейших этапов разработки и тестирования, немаловажным критерием является здравый смысл и жизненный опыт команды реализации проекта, позволяющие избежать будущих трудностей по причине человеческого фактора. Например, в спецификации описана функция загрузки файлов пользователя в программу с ограничением до трех файлов за раз. Предположим, что пользователю нужно загрузить сотню файлов и тогда ему придется повторять пройденные действия множество раз, что приведет к недовольству пользователя и неудовлетворительному результату обеспечения качества и удобства использования программы. В таких случаях специалистам по обеспечению качества следует оповестить заказчика программы и отправить разработчикам запрос на улучшение или изменение (Improvement или Change Request[4]) функциональности программы.

[4] Improvement – это тип запроса команде разработки проекта и бизнес-аналитикам, направленный на улучшение функции или логики проектируемого программного обеспечения.

В процессе подготовки и реализации процесса тестирования не следует забывать про человеческий фактор риска формирования дефектов спецификации. Любой человек по своей природе склонен совершать ошибки, которые сложно вовремя заметить. При разработке технической документации как ожидаемого результата работы программы необходимо проводить дальнейший анализ по определенным критериям: однозначность, полезность, тестируемость, понятность, завершенность, выполнимость, непротиворечивость, независимость [22]. Разберем отрывок из спецификации веб-сайта и проведем анализ:

«Пункт 15.1 спецификации “Об отправке уведомлений на электронную почту” описывает: Поле “Ваш адрес электронной почты” должно быть обязательным для заполнения. Уведомление об ошибке должно отображаться в случае, если пользователь не заполняет указанное поле.».

В соответствии с данным пунктом спецификации специалисту разработки дается свобода в принятии решения о тексте ошибки и есть возможность прописать в коде программы несколько вариантов:

– программа отобразит неинформативное сообщение об ошибке «Ошибка» и пользователь, столкнувшийся в первый раз с данной программой, вынужден искать самостоятельно причину уведомления об ошибке;

– программа выведет информативное сообщение «Введите ваш адрес электронной почты в поле».

Оба вышеперечисленных варианта будут формально верны в соответствии с текстом спецификации, так как конкретное требование к тексту ошибки не описано. Однако в процессе тестирования данной функции специалист по обеспечению качества столкнется с вопросами «какой должен быть текст ошибки как ожидаемый результат» и «так ли хотел бы увидеть заказчик текст ошибки». Таким образом, пункт в спецификации не соответствует критериям однозначности, завершенности и понятности, и, в

Change Request – это тип запроса команде разработки проекта и бизнес-аналитикам, направленный на внесение необходимых изменений в проектируемое программное обеспечение.

свою очередь, нуждается в дополнительной доработке бизнес-аналитиками и заказчиком проекта.

Итак, с целью обеспечения качества конечного продукта команда специалистов в области тестирования должна привлекаться на всех этапах жизненного цикла проекта и находить дефекты не только уже разработанного кода, но и технической документации.

На этапе планирования тестирования от команды обеспечения качества требуется владение практическими и теоретическими основами в профессиональной области, а также немаловажен творческий подход, который позволит предусмотреть самое безумное обращение будущего пользователя с программой. Принципы найденных решений планирования этапа тестирования заключаются:

– в простоте, краткости и изящности сценариев проверки функциональной части проекта;

– в обеспечении компромисса и разумности между объемом тестовых сценариев, возможных в теории, и объемом сценариев, допустимых с практической точки зрения.

Для подготовки к этапу тестирования команде необходимо продумать не только сценарии проверки каждого реализованного компонента программы, но и необходимые предварительные условия и требования к началу реализации сценария.

Рассмотрим пример программы, разработанной для компании в области телекоммуникаций. Одним из возможных сценариев является следующий:

– войти в программу под логином «test@test.com» и паролем «Test»;

– добавить в корзину тариф «Классический»;

– выбрать доступный номер телефона из каталога;

– оплатить тариф.

В данном случае пункты 1-4 включительно выступают как подготовленный сценарий проверки «Покупка тарифа «Классический»». В первом пункте указанные данные являются предварительным условием, так

как сначала необходимо зарегистрировать пользователя с вышеуказанными логином и паролем. Для проверки третьего пункта сценария важно подготовить несколько вариантов телефонных номеров, доступных для выбора. Таким образом, предварительные данные или условия – это фиксированные значения ввода с определенными критериями, необходимые для прохождения тестирования и достижения фактического результата.

Исполнение тестирования подразумевает практическое прохождение подготовленных сценариев и поиск дефектов в реализованном программном обеспечении.

Выделяют две распространенные концепции проведения тестирования программного обеспечения, приводящие, в частности, команду обеспечения качества к ложному направлению. [21]

Первая концепция заключается в достижении цели тестирования как проведения проверки программы от самого первого символа кода до последнего. Данная концепция приводит к необъятному объему работы команды и отсутствию необходимого результата. Рассмотрим пример программы, написанной на языке Python[5]:

user input = raw_input (“Type your favorite social media”)

if raw_input == “Whatsup”: print “green icon”

else if raw_input == “VK”: print “blue icon”

else if raw_input == “Instagram”: print “red icon”

В данном случае программа является очень узконаправленной и бессмысленной, судя по которой можно выделить только три условия ввода значений пользователем, запускающих успешную работу программы. Следует учитывать дополнительное условие, не указанное в самой программе, при котором пользователь вводит неверное значение или не вводит какое-либо значение. Таким образом, для тестирования вышеуказанной программы

[5] Python – это распространенный объектно-ориентированный язык программирования со строгим читаемым синтаксисом кода, ориентированным на повышение производительности разработчика и переносимость написанных программ на различные окружения.

необходимо провести четыре сценария для проверки работоспособности всех условий программы. В подобных случаях существует два пути ввода значений, таких как действительные и недействительные данные. Действительным вводом данных являются значения, предусмотренные логикой разработанной по требованиям программы и приводящие к ожидаемому результату заказчика. К недействительному вводу данных относятся значения, не упомянутые в коде программы и не указанные в технической документации проекта. Вышеупомянутый пример ввода пустого значения является как действительным, так и недействительным, так как условия обработки такого ввода могут быть утверждены или не утверждены в технической документации.

Итак, в примере программы, написанной на языке Python, для тестирования достаточно было бы пройти всего четыре сценария, однако в реальных условиях разработки подобных условий может быть сотни тысяч. Например, если для одного параметра пользователю допустим ввод значения от 1 до 999999. Проверка каждого условия такой программы приведет к значительным финансовым затратам со стороны заказчика, увеличению сроков реализации проекта и пустой трате человеческих ресурсов.

Таким образом, следование сущности первой концепции неоправданно для реальных проектов с ограниченными ресурсами, профессиональной команде обеспечения качества необходимо применять творческий подход к планированию и реализации тестирования программного обеспечения.

Идея второй концепции заложена в нахождении наибольшего количества дефектов программного обеспечения как критерия эффективности тестирования.

Для доказательства ложности данного суждения рассмотрим пример приготовления авторского меню для предприятия общественного питания. Покупателя в конечном итоге не будет интересовать время приготовления блюда и стоимость всех ингредиентов, только качество и вкус блюда, а также конечная цена за порцию. Если предприятие будет продавать блюда

авторского меню за неприлично высокую стоимость, то покупатели скорее всего будут рассматривать другие предприятия-конкуренты. Таким образом, специалистам в области тестирования правильно будет воспринимать безразличие пользователя к количеству затраченных ресурсов как естественное поведение, так как пользователь нуждается только в фактически полученным качественным продуктом.

Таким образом, процесс тестирования является важным этапом жизненного цикла разработки программного обеспечения и направлено на нахождение дефектов. Целью обеспечения качества продукта, в которое вовлечены все участники проекта, является предотвращение потенциальных дефектов на ранних этапах разработки. Опытный специалист по обеспечению качества не руководствуется ложными концепциями тестирования программного обеспечения и следует рациональному и логичному плану тестирования.

1.3 Классификация методов тестирования программного обеспечения

Под методом тестирования программного обеспечения подразумевается особый метод исследования и проверки программного обеспечения на соответствие разработанной технической документации. Методы тестирования классифицируются по определенному признаку, рассмотрим основные критерии:

- по объекту тестирования;
- по субъекту тестирования;
- по знанию внутренней структуры системы и кода;
- по степени автоматизации тестирования внешними системами;
- по достижению определенного этапа разработки программы;
- по критерию соответствия ожидаемому поведению программы.

К методам тестирования по объектной классификации относится тестирование интерфейса пользователя, безопасности, совместимости, скорости и надежности, локализации, удобства пользователя, функциональное тестирование [12].

Под тестированием пользовательского интерфейса подразумевается проверка работоспособности элементов интерфейса пользователя как описано в требовании спецификации. Например, требование описывает название поля «Имя» и ограничивает максимальное количество ввода символов до тридцати. Соответственно тестирование интерфейса будет заключаться в проверке названия поля «Имя» (а не «Name») и вводе более тридцати символов в поле.

Тестирование множественных аспектов, связанных с адаптацией программы для пользователей, которые находятся на территориях разных стран и говорят на соответствующих языках, является тестированием локализации. Предположим, что пользователь при регистрации в программе выбрал Испанию как фактическое место проживания и испанский язык как родной. Тестирование локализации будет заключаться в проверке перевода всех кнопок и текста программы на испанский язык и выводе сообщения об ошибке в случае, если данный пользователь введет свои данные раскладкой и шрифтом на другом языке.

Тестирование скорости и надежности реализуется для проверки поведения реализованного программного обеспечения при одновременной нагрузке множества пользователей. Целью данного метода тестирования является поиск и нахождение слабых мест программного обеспечения. Примером реализации тестирования скорости и надежности может быть проверка времени ожидания загрузки страниц программы или значений базы данных путем использования специальных систем для имитации нагрузки пользователями. Медленная скорость реакции программы на действия пользователя может привести к потере потенциальных реальных пользователей.

Следующим методом объектного тестирования является тестирование безопасности. В современном мире информационных технологий формируется тенденция развития криминальной индустрии в сети Интернет. Тестированием безопасности занимается специальная команда, цель которой поставить себя на место мошенников, провести ряд манипуляций со взломом и в результате усложнить условия взлома системы безопасности разрабатываемой программы.

Тестирование удобства пользователя направлено на оценку опыта потенциального пользователя будущей программы, действия которого должны иметь возможность осуществляться интуитивно. Данный метод тестирования реализуется благодаря привлечению группы пользователей, не знакомых с программой, для сбора комментариев по удобству созданного интерфейса. Например, пользователь ожидает увидеть кнопку оплаты товара в правом нижнем углу интерфейса корзины с товаром и, если данная кнопку находится в другом расположении, пользователь вынужден провести анализ всей страницы интерфейса.

При тестировании совместимости как объектного метода тестирования проводится проверка взаимодействия созданной программы с другими системами, например, с браузерами (cross-browser testing[6]) или операционными системами (cross-platform testing[7]) на компьютерах потенциальных пользователей. Стоит отметить как пример созданный текстовый редактор Microsoft Word, который был реализован только для использования в операционной системе Windows. Соответственно, при попытке установить данную программу в системе IOS возникают трудности и даже при успешной установке неофициальной версии редактора пользователь столкнется с трудностями и критичными ограничениями в работе с редактором.

[6]Cross-browser testing – это метод тестирования совместимости программы с различными браузерами (Chrome, Yandex, Microsoft Edge и тому подобное).

[7] Cross-platform testing – это метод тестирования совместимости программы с разными операционными системами (Linux, IOS, Windows и тому подобное).

Функциональное тестирование заключается в процессе проверки работоспособности каждой реализуемой функциональности программного обеспечения в соответствии с требованиями, описанными в спецификации [27]. Примером функционального тестирования может послужить следующий случай:

«Пункт 9.03 спецификации “О регистрации нового пользователя” предполагает: поля “Имя”, “Адрес электронной почты”, “Номер телефона” и “Пароль” являются обязательными для заполнения. При не заполнении какого-то из полей появится ошибка “Поле ___ не заполнено”. При заполнении всех вышеперечисленных полей после нажатия кнопки “Зарегистрироваться” пользователь получает доступ к программе как клиент. Личная информация о пользователе сохраняется в специальную таблицу базы данных.».

Для реализации функционального тестирования данного отрывка спецификации необходимо проверить четыре возможных сценария, каждый из которых фиксирует работоспособность отдельной функциональности:

– при пропуске одного из полей, обязательных для заполнения, пользователь получает ошибку о не заполненном поле;

– при пропуске одного из полей, обязательных для заполнения, пользователь не может получить доступ к программе;

– при заполнении всех обязательных полей пользователь заходит в программу как клиент;

– при регистрации пользователя личные данные заносятся в таблицу базы данных.

Методы тестирования программного обеспечения по субъекту делятся на альфа-тестирование и бета-тестирование. При использовании первого метода участвует сотрудник компании со свежим взглядом, который ранее не принимал участие в поиске дефектов программы, накануне выдачи проекта заказчику. Для проведения второго метода тестирования IT-компания привлекает сторонних людей, которые не являются участниками проекта, и дает возможность попробовать новую программу [14].

По знанию внутренней структуры системы выделяют такие методы тестирования программного обеспечения, как тестирование «черного ящика», «серого ящика» и «белого ящика» [21].

Тестирование методом «черного ящика» подразумевает отсутствие знаний о внутреннем коде и структуре системы у специалиста, проходящего сценарий тестирования.

С одной стороны, в данном случае команда отдела тестирования обладает преимуществом над создателями программы. Предположим, что на этапе разработки кода программы команда разработчиков проводила свою проверку работоспособности программы, однако в процессе проверки автор кода мог пропустить простую часть кода, так как сам был уверен в исправности логики программы и в соответствии со спецификацией. Разработчик точно знает каким образом в его понимании должна работать программа и он не может предусмотреть возможные сценарии развития нелогичных по коду действий пользователя при работе в программе. Нередко в подобных ситуациях на этапе тестирования обнаруживаются ошибки именно теми специалистами, которые не знают внутреннюю логику кода. С другой стороны, незнание команды обеспечения качества о внутренней структуре кода программы может привести к увеличению сроков на проверку одного компонента или функции. Например, одну функциональность программы достаточно проверить двумя сценариями, однако команда тестирования «вслепую» выполняет более десяти сценариев.

Вторым признаком тестирования методом «черного ящика» является реализация идей тестирования на основе предполагаемых шаблонов поведения потенциального пользователя. Источниками данных идей обычно могут быть спецификации, спонтанные исследовательские действия в программе, интуитивные сценарии достижения ожидаемого результата (например, оформить заказ мебели любым доступным способом), результаты консультации с командой разработки и бизнес-аналитиками и опыт самого специалиста в области тестирования.

При использовании метода «белого ящика» специалист по обеспечению качества, в отличии от метода «черного ящика», проводит проверку на основе имеющихся полноценных знаний о логике и структуре кода программы. В данном случае идеи команды тестирования направлены на проверку конкретной части кода вне зависимости от шаблонного поведения пользователя.

Применение методов «черного ящика» и «белого ящика» одновременно для тестирования программы позволяет количественно и качественно увеличить покрытие возможных сценариев при помощи разных точек зрения.

Выделяют третий метод тестирования «серый ящик», основанный на смешанном сочетании идей «черного ящика» и «белого ящика». Рассмотрим пример тестирования вышеупомянутого функционала, описанного в спецификации “О регистрации нового пользователя”. Предположим, что при проверке следующего сценария пользователь получил доступ к программе:

– открыть программу;
– заполнить все обязательные поля;
– нажать кнопку «Зарегистрироваться».

Полученный доступ не может быть точным подтверждением успешного фактического результата, так как пользователь даже не получает уведомление об успешной регистрации. Необходимо провести на основе текста спецификации дополнительную проверку, состоящую из поиска новой записи о зарегистрированном пользователе в базе данных [23]. Таким образом, при проведении анализа спецификации специалист по обеспечению качества может выявить ожидаемое поведение работы программы как в пользовательском интерфейсе, так и в коде [15].

К следующей группировке методов тестирования по степени автоматизации тестирования внешними системами относятся ручное, автоматизированное и смешанное (полуавтоматизированное) тестирование [24].

К ручному тестированию относится подготовка предварительных данных и исполнение сценариев без привлечения каких-либо автоматизирующих программ [25]. Например, для выполнения сценария по покупке тарифа поддержки мобильной связи в телекоммуникационной компании необходимо вручную сначала отрыть страницу регистрации и создать нового пользователя.

Под методом автоматизированного тестирования подразумевается привлечение внешних инструментов автоматизации для полного покрытия всех возможных сценариев тестирования. Автоматизацией называется комплекс мероприятий, направленный на повышение производительности труда человека посредством замены данного труда работой машин. Наиболее частое использование инструментов автоматизации встречается на долгосрочных проектах с регулярной выдачей новой версии программы заказчику [18]. Автоматизированные тестовые сценарии позволяют воспроизводить ручные действия пользователя и сравнивать ожидаемый и фактический результат. Данный метод тестирования дает возможность команде по обеспечению качества ускорить процесс тестирования новой версии программы перед выдачей заказчику и уменьшить количество сотрудников, привлеченных к тестированию. Различные IT-компании при поддержке долгосрочного проекта разрабатывают наборы автоматизированных тестовых сценариев, применяя наиболее удобный способ (записать конкретные действия пользователя и преобразовать их в код, или непосредственно написать код тестового сценария), и периодически запускают их на тестовой версии программы. Автоматизированный тестовый сценарий также называется «автотест» в профессиональной сфере и является скриптом, имитирующим взаимодействия пользователя с программным обеспечением и направленным на локализацию ошибок в работе программы.

Смешанное (полуавтоматизированное) тестирование выполняется в виде сочетания ручного и автоматизированного методов тестирования. Например, для создания предварительных данных применяется

автоматизированная внешняя система, а для дальнейшего прохождения сценария используется ручной подход.

По критерию соответствия ожидаемому поведению программного обеспечения выделяются такие методы тестирования как позитивное и негативное тестирование.

Негативный метод тестирования программы заключается в исполнении таких сценариев, цель которых направлена на проверку потенциальной ошибки в работы пользователя и на поиск возможных дефектов в логике самой программы. Примером ошибки пользователя может выступать неверный ввод значений в поля для регистрации, а ошибкой программы является неправильное внесение значений пользователя в таблицы базы данных.

Позитивное тестирование предполагает проверку программы по сценариям, направленным на нормальное поведение программы и ожидаемую работу пользователя в ней. Позитивным сценарием проверки работы в программе, например, является ввод действительных значений в поле для регистрации, а сценарий проверки поведения программы заключается в тестировании добавленного пользователя в базу данных. Как правило, позитивное тестирование необходимо для проверки работоспособности основных функций программы в соответствии со спецификацией и поэтому проводится в первую очередь. Затем инициируется негативное тестирование программы для поиска непокрытых спецификацией и решением ошибок и дефектов.

Разделяют методы тестирования по достижению определенного этапа разработки программы: приемочное тестирование, альфа-тестирование, бета-тестирование [21].

Приемочное тестирование реализуется в промежутке между этапами разработки кода программы и тестирования. Разработчики программы завершают свои доработки кода и передают специалистам по обеспечению качества тестовый вариант программы. В первую очередь, команда тестирования проверяет главные функции программы на блокирующие

дефекты. Например, если пользователь не может получить доступ к программе после регистрации, то дальнейшее тестирование невозможно и программа возвращается на исправление и доработку команде разработки.

После проведения запланированного тестирования наступает этап реализации альфа-тестирования, который осуществляется перед выдачей программы потенциальным пользователям заказчика. При успешном результате следующим этапом инициируется бета-тестирование, для которого привлекается ограниченная группа пользователей из целевой аудитории программы. При бета-тестировании такие пользователи называются «бета-тестировщики», а версия программы – «бета-версия».

Принципиальное отличие альфа-тестирования и бета-тестирования заключается в привлеченных ресурсах, то есть для альфа-тестирования привлекают сотрудников IT-компании, участвовавших в предыдущих этапах разработки программы, а для бета-тестирования используют внешние ресурсы как потенциальных пользователей, так и окружения для установки бета-версии.

Таким образом, в настоящее время выделяют разнообразные методы обеспечения качества проекта и максимального покрытия решений. Для выполнения дальнейшего исследования достаточно вышерассмотренных теоретических основ распространенных методов тестирования программного обеспечения, информации об их особенностях, назначении и области применения. Главная цель каждого из подходов заключается в поиске и устранении найденных дефектов до того, как их обнаружат потенциальные пользователи. Следует учитывать предусмотренную последовательность реализации методов тестирования, которая позволит максимально сократить и предотвратить дальнейшие проблемы, влияющие на стоимость, трудоемкость и сроки выполнение проекта.

Глава 2 Практические методы обеспечения качества программного обеспечения

2.1 Влияние технической документации на обеспечение качества программного продукта

Разработка и реализация программного обеспечения подразумевает исполнение всех этапов цикла разработки, одним из которых, в первую очередь, выступает формирование необходимой документации по проекту. Каждый этап разработки программы должен выполняться в определенные сроки и с конкретной последовательностью.

Важную роль в разработке и обеспечении качества проекта играют спецификации и дизайны. Спецификация представляет собой документ, в котором собраны детальные описания каждого модуля и каждой функции программы. Пользовательский интерфейс планируется также на первых этапах жизненного цикла и формируется в документах дизайна. В дизайне описываются расположения модулей, кнопок, ссылок, картинок, а также их определенные сочетания цветов и размеров, которые должны быть созданы в конечном итоге проекта. На этапе разработки программисты, как правило, берут за основу спецификации и дизайны для написания кода программы и создания пользовательского интерфейса, а на этапе тестирования специалисты по обеспечению качества сравнивают фактически созданную программу с документами, указывают на расхождения документации и программы и добиваются максимального совпадения. Вышеупомянутая документация формируется по окончании процесса разработки требований [5].

Разработка требований к программному обеспечению — это процесс, состоящий из мероприятий, необходимых для создания и утверждения документов, содержащих спецификацию и дизайн системных, функциональных, пользовательских требований, а также бизнес-требований заказчика проекта [9]. Выделяют четыре основных этапа процесса разработки

требований, взаимосвязи между которыми и соответствующие документы изображены на рисунке 3:

– анализ технической осуществимости создания проекта;

– формирование и анализ требований;

– специфицирование требований и создание соответствующей документации;

– аттестация собранных требований.

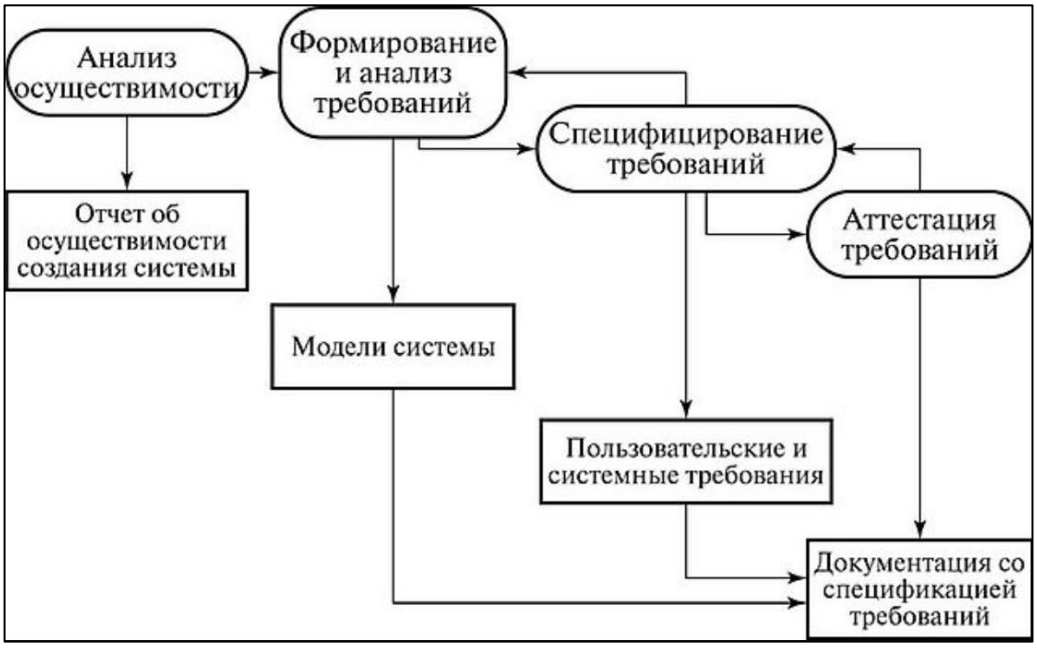


Рисунок 3 – Процесс создания требований к программному обеспечению

Под первым этапом подразумевается анализ осуществимости требований, который начинается с общего описания программы и её назначения и результатом которого выступает отчёт с чёткой рекомендацией о продолжении или прекращении дальнейших шагов разработки требований. Анализ осуществимости раскрывает вопросы о соответствии проекта бизнес-целям заказчика и команды разработки, о возможности реализации задуманного проекта в рамках установленного бюджета и срока, а также о

возможности интеграции планируемого проекта с уже существующими программами у заказчика.

Для реализации следующего этапа необходимо применять различные практики и подходы к установлению источников требований и извлечению требований [3]. С целью построения доступной для понимания структуры требований, которая будет отвечать осуществимым запросам и приоритетам заказчика, возможно применение следующих методов:

– проведение интервьюирования заказчика, в котором полученная информация будет являться «входом» для процесса сбора требований, а выделенные в результате требования заказчика будут выступать «выходом» процесса;

– формирование сценариев для сбора пользовательских требований, которые будут раскрывать основные функции программы и нестандартные случаи в отношении бизнес-процессов, выполняемых пользователем;

– создание такого инструмента для уточнения и детализации требований, как прототип, визуализирующий сценарии или выполняющий основные функции;

– организация встреч заинтересованных лиц, которые планируются с определенной периодичностью, с целью обсуждения спорных вопросов и доведения информации до однозначности [19];

– установка наблюдения бизнес-аналитиков и инженеров над процессом выполнения пользователем бизнес-процессов.

Планируемая спецификация обладает рядом последовательных статусов подготовки. В процессе формирования и структуризации требований спецификациям и дизайнам присваивается статус «Черновика» (Draft) [21].

Следующим этапом процесса разработки требований является формирование и анализ требований. Данный этап необходим для выявления области применения, описания разрабатываемой программы, режима её работы, параметров выполнения её функций и установок ограничений. Этап формирования и анализа требований отличается своей цикличностью,

реализованной от анализа предметной области до проверки требований, и наличием обратной связи взаимодействующих между собой этапов процесса разработки требований. Анализ требований не производится по универсальному подходу, однако подразумевает проверку по определенным показателям:

– корректность требований, отсутствие ошибок в формулировке задачи;

– однозначность, при которой команда разработки не совершит ошибку в толковании требования;

– отсутствие противоречивых друг другу требований;

– полноценный набор требований, описывающий каждую деталь разрабатываемого проекта;

– проверяемость каждого требования для возможности дальнейшего тестирования;

– понимаемость требований, то есть описание требования реализовано на понятном языке для каждого участника процесса разработки и реализации проекта.

Рассмотрим пример требования, в котором присутствуют несколько отклонений по вышеупомянутым показателям. Требование звучит: «Обязательное поле “номер телефона” должно заполняться девятью цифрами. В случае набора букв или большего количества цифр должно появиться окно с ошибкой.». В данном случае не упомянут текст сообщения с ошибкой, который должен уведомить пользователя, а также не указано ожидаемое поведение программы в случае заполнения поля пунктуационными или математическими знаками. При проведении анализа данного требования должны быть учтены отклонения по нескольким показателям: корректности требования и полноценности набора требований. Недостающую и неоднозначную информацию необходимо обсудить с заказчиком и дополнить требование.

В приведенном выше примере присутствует дефект в функциональной спецификации. Решением сложившейся проблемы послужат мероприятия выявления дополнительной информации: обсуждение и утверждение определенного текста ошибки, который позволит пользователю понять какое именно его действие привело к ошибке; разработка сценария реакции программы в случае отклонения пользователя от ожидаемых действий. При невыполнении мероприятий по уточнению требований возможно упущение дальнейших дефектов программы. С каждым последующим этапом разработки проекта упущенные дефекты требуют еще больших затрат из бюджета, рассчитанного на реализацию проекта, и большего количества времени на устранение.

Во избежание проблем в процессе разработки и реализации проекта инициируется следующий этап процесса создания требований к проекту, а именно специфицирование требований. На данном этапе документируется соглашение заказчика и команды разработки проекта, содержащее бизнес-требования к проекту и возможные сценарии работы пользователей в запланированной программе. Для согласования и утверждения спецификации и дизайна необходимо структурировать собранную информацию, сформировать её в доступном виде для всех заинтересованных лиц, ознакомить с документами команду разработки и заказчика с целью получения согласия со всеми пунктами требований. В случае необходимости документация дополняется иллюстрациями, графическими моделями, таблицами, четкими и понятными формулами для наиболее наглядного представления программы, её изменения при взаимодействии с пользователем и вводимыми данными, взаимозависимости классов объектов программы и отношений между ними.

Спецификации и дизайны, прошедшие вышеупомянутый этап обработки, приобретают следующий статус перед утверждением финальной версии документов «Ожидание утверждения» (Approval Pending) [21].

Для утверждения спецификации и дизайна будущего проекта должна быть организована встреча заказчика и команды разработки. При получении положительной реакции всех заинтересованных лиц и решении спорных требований документация получает статус «Утверждено» (Approved) [21].

Таким образом, на всех последующих этапах разработки и реализации проекта работа специалистов основана на созданной технической документации. Данные документы играют важнейшую роль жизненного цикла проекта, так как в них описана и структурирована вся необходимая информация по ожидаемому программному обеспечению, его функциональности, назначению и интерфейсу пользователя.

Для систематизации процесса обеспечения качества необходимо формирование дополнительной документации, такой как план и стратегия дальнейшего тестирования «сырого» проекта.

Стратегия тестирования описывает область программного обеспечения, подлежащая тестированию, подход к тестированию, основные этапы и методы тестирования, а также определяет необходимые инструменты, требования к тестовому окружению и приоритеты результатов исполнения тестирования.

План тестирования представляет собой документ, обобщающий и координирующий тестирование программного обеспечения, содержащий в себе важные выдержки из спецификации по разработке программного обеспечения с целью более эффективного обеспечения качества [4]. Составление данного документа позволяет определять временные рамки тестирования программного обеспечения и степень нагрузки задачами каждого специалиста по обеспечению качества, а также сфокусировать внимание на первоначальной цели разработки программного обеспечения, что в результате приводит к повышению вероятности успешного завершения тестирования.

В цикле разработки проекта этап тестирования также разбивается на этапы, на каждый из которых назначается статус выполнения (открыт, повторно открыт, в процессе, разрешен, закрыт) [21]. Статус позволяет

формировать регулярную отчетность прохождения тестирования и регулировать этапы. Смена статуса зависит от количества и сложности найденных ошибок и отклонений от технических требований. Соответственно на каждом этапе тестирования необходимо сверять текущую деятельность с ожидаемой, используя документ «план тестирования». Детальное описание функциональных и нефункциональных требований, бизнес-требований и дизайна пользовательского интерфейса содержится в техническом документе – спецификации. На основе спецификации специалист по обеспечению качества сверяет фактическое состояние разработанной программы с ожидаемым, а в случае расхождения фиксирует дефект, который передается специалисту разработки соответствующего модуля с подробным описанием и ссылкой на определенный текст из спецификации. У каждого найденного дефекта также существует жизненный цикл, изображенный на рисунке 2, который характеризуется различными статусами выполнения корректировки.

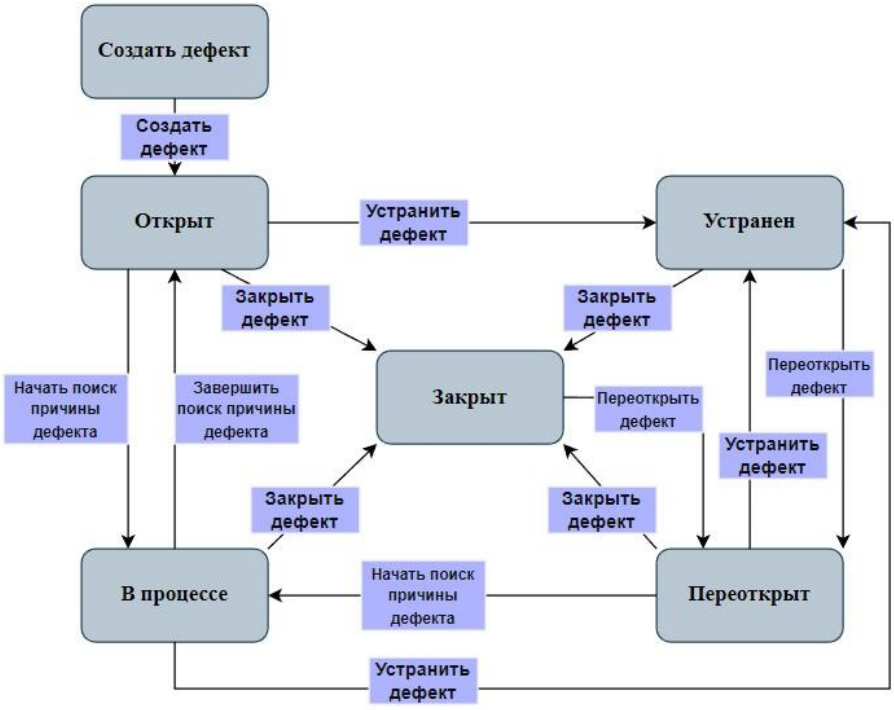


Рис. 2. Жизненный цикл дефектов программного обеспечения

Рассмотрим структуру плана тестирования и его содержимое в деталях. План тестирования состоит из следующих разделов:

– введение (Introduction) – содержит общую информацию о проекте и заказчике;

– область тестирования – описывает конкретные тестируемые компоненты, функции и тестовые данные, которые определены заказчиком. Формируются тестовые кейсы для покрытия всех возможных сценариев;

– процесс тестирования программы – распределяются роли сотрудников и их обязанности. Ставятся первоначальные требования перед тестированием кейсов и шаги с предварительными условиями для запуска тестирования. Руководитель отдела тестирования расписывает сроки проведения функционального, нефункционального и приемочного тестирования, а также составляет расписание выполнения каждого тестового кейса. Определяются критерии приостановки и возобновления тестирования, описываются необходимые виды отчетности о пройденном тестировании.

Для более детальной демонстрации представим тестирование программы для сбора заказов и формирования чеков клиентам кафе. При функциональном тестировании примером тестового сценария послужит «Добавление блюда в заказ клиента». Ожидаемым результатом является два добавленных блюда в заказ с верным подсчетом чека. К нефункциональному тестированию можно отнести сценарий «Пользователю удобно отправлять готовую форму заказа». Комфортный интерфейс для пользователя без видимых дефектов и сбоев работы программы является ожидаемым результатом данного сценария. В приемочном тестировании проходят тестовые сценарии на основе основных бизнес-требований заказчика. Например, «Чек клиента успешно отправляется в работу» основан на проверке работоспособности главной цели программы, что и есть ожидаемый результат данного тестового сценария.

– процесс приемочного тестирования – формируется таблица по первоначальным требованиям с предварительными условиями для начала

приемочного тестирования. Цель данного этапа заключается в окончательной проверке проекта заказчиком, который оценивает фактическую готовую программу по результатам прохождения тестовых сценариев, затрагивающих наиболее важные бизнес-требования заказчика.

Рассмотрим более детально второй раздел плана тестирования.

Тестовый сценарий, в профессиональной сфере называемый тест-кейсом, характеризуется как сравнение фактического результата с ожидаемым. Структура тест-кейса состоит из инструкции по вводу (шаги), исполнении шагов (ввод), ожидаемого результата (ожидаемый ввод) и фактического результата (фактический ввод) [28]. Исход тест-кейса может быть положительным в случае совпадения фактически полученного результата с ожидаемым, в ином случае (при отклонении от ожидаемого результата) – отрицательным. Таким образом, отрицательный исход прохождения тестового кейса является дефектов, также называемым в профессиональной сфере багом[8].

Основными причинами появления дефектов в процессе тестирования программы могут выступать следующие случаи [21]:

– непонимание или отсутствие достаточно четкой информации либо у специалиста по разработке в процессе написания кода программы, либо у специалиста по обеспечению качества при тестировании программы;

– высокая сложность взаимосвязи и содержимого модулей и функциональностей программы;

– изменения требований заказчика или введение новых функциональностей на сложных этапах разработки проекта;

– недостаточно грамотный документированный код специалистов разработки программы;

– трудности использования средств и программ разработки проекта.

[8] Баг – это отклонение фактически полученного результата прохождения тест-кейса в реализуемой программе от ожидаемого результата, описание которого формируется на основе технической документации и бизнес-требований заказчика.

Обеспечение качества проекта характеризуется как «забота» о качестве через улучшение процесса разработки программы в виде превентирования появления багов на каждом этапе, в различных источниках данный подход называется Quality Assurance. В свою очередь, процесс тестирования направлен на обнаружение багов до передачи готового релиза проекта заказчику и выдачи пользователям.

Появление багов возможно на всех этапах разработки проекта, в первую очередь, на этапах формирования требований к проекту как функциональных, так и бизнес-требований. За своевременным контролем и устранением дефектов следят бизнес-аналитики, руководитель проекта, руководители отдела разработки и отдела тестирования. Устранение ошибок на ранних этапах разработки проекта позволяет значительно сэкономить финансовые ресурсы и укоротить сроки реализации проекта.

Специалисты по разработке на этапе кодирования также выполняют тестирование главных модулей и функций проекта при помощи запуска определенной части кода через инструменты разработки. Данный процесс называется юнит тестирование (unit test) или модульный тест, который позволяет выявить основные важнейшие дефекты проекта при внесении изменений в код или добавлении новых частей кода.

Наибольший акцент на нахождение багов реализуется на этапе тестирования программы при помощи тест-кейсов, рассчитанных на покрытие всех возможных решений. Обнаружение огромного количества багов не гарантирует высокое качество проекта, однако выявление и устранение блокирующих и масштабных багов необходимо для работоспособности программы.

Таким образом, техническая документация проекта является информационной базой для команд разработки и тестирования и представляет собой неотъемлемую часть процесса обеспечения качества разрабатываемой программы. Дефектом программного обеспечения является несоответствие фактического результата с ожидаемым, который описывается в спецификации

и основывается на бизнес-требованиях заказчика. Обнаружение бага возможно на любом этапе жизненного цикла разработки проекта и источники появления дефектов могут зависеть как от технических причин, так и от человеческого фактора. Каждый участник проекта несет ответственность за обеспечение качества разработки вне зависимости от масштаба проекта, количества ресурсов или сроков выполнения. Для планирования тестирования, в первую очередь, составляется стратегия для координации дальнейших действий команды по обеспечению качества. План тестирования, разработанный на первых этапах жизненного цикла проекта, позволяет команде ориентироваться на расписание реализации тестовых сценариев и их шаги, предусматривать возможные риски найденных ошибок и составлять на основе установленных требований отчетную документацию по завершению каждого этапа тестирования.

2.2 Классификация дефектов релиза и процесс их устранения

Одним из общепринятых определений дефекта программного обеспечения является расхождение между фактически готовым проектом и его спецификацией, только при условии, что спецификация существует и её содержимое сформулировано детально и правильно [21].

Основным критерием при определении бага выступает наличие или отсутствие реализации проектом требований, которые ожидают получить пользователи. Соответственно при отсутствии реализации фиксируется дефект программного обеспечения, который необходимо устранить.

Рассмотрим существующие классификации багов программного обеспечения в зависимости от источника выявленной и проанализированной проблемы.

В зависимости от степени автоматизации тестирования программного обеспечения могут быть найдены различные виды дефектов. В настоящее время для средних и масштабных проектов используется как ручное, так и

автоматизированное тестирование. Ручное тестирование (manual testing) происходит без помощи программ, автоматизирующих работу и заменяющих действия специалистов по обеспечению качества. Автоматизированное тестирование (automated testing) запускается благодаря специальным инструментам и компьютерным программам со вспомогательными функциями и свойствами. К подобным видам автоматизации относятся программы для регрессивного тестирования, программы для тестирования скорости и надежности, программы для поддержки тестирования черного и серого ящика [13].

В процессе реализации ручного тестирования возможно обнаружение следующих дефектов программного обеспечения:

– environment issue – это дефект, основанный на несовместимости тестового окружения с установленными новыми настройками или билдами[9], или дефект на основе неправильных настроек и компиляции кода в билде;

– network issue – это баг, вызванный из-за разрыва соединений сети, поддерживающей взаимосвязь модулей проекта, базы данных, системы управления базами данных, сервера и других элементов, необходимых для стабильной интегрированной работоспособности проекта;

– configuration issue – дефект на основе неверно заданных пользователем или специалистом разработки совокупностей настроек программы, которые являются конфигурацией программного обеспечения;

– performance issue – это ошибка, возникающая в большинстве случаев при нагрузочном тестировании, когда система не выдерживает или не рассчитана на одновременное выполнение множества действий и запусков функций;

– test data issue – это баг, основанный на создании ошибочных предварительных условий или данных, на реализации неверных действий,

[9] Build - конечный результат компиляции программы с уникальным номером версии сборки, планируемой к выдаче заказчику в момент релиза.

которые влияют на дальнейшее тестирование, но не приводят к ожидаемому результату;

– solution issue – это основной искомый баг, который возникает по причине некорректно составленного кода и сформированного решения проекта. Данный баг может повлиять на результаты вычислений, запуски важных функций и выгрузки отчетов, а также может заблокировать связанные с них действия в программном обеспечении;

– issue with design change – дефект, реализуемый при изменении бизнес-требований или требований к программному обеспечению, которые в дальнейшем формируются и вносятся в спецификации и дизайны проекта.

Для реализации ручного тестирования, как правило, используются заранее подготовленные тестовые сценарии с полным покрытием всех очевидных решений программного обеспечения и привлекаются специалисты по обеспечению качества, каждый из которых несет ответственность за определенный набор тест-кейсов.

Для предотвращения ошибок, связанных с тестовых окружением, специалисты по обеспечению качества, в первую очередь, проверяют совместимость тестируемых модулей программного обеспечения, настройки конфигурации, а также фиксируют устойчивое соединение с сетью [10]. Таким образом, создаются благоприятные условия для дальнейшего прохождения ручного тестирования без потери времени на такие дефекты, как environment issue, network issue, configuration issue.

При запуске автоматизированного тестирования через определенные программы можно встретить все вышеперечисленные дефекты, а также такие виды багов, как:

– AT[10] issue – ошибка, которая вызвана неправильной согласованностью шагов автоматизированных тест-кейсов, составленных

[10] AT (англ. automated testing) – аббревиатура, обозначающая в данном контексте автоматизированное тестирование.

специалистами по обеспечению качества или специалистами по автоматизированному тестированию;

– AT Maintenance – баг системы для проведения автоматизированного тестирования программы, который возникает с случаях перебоя сети в данной системе или запланированной установкой новой версии;

– AT Warning – не является ни багом, ни показателем успешного прохождения тест-кейса. Данное уведомление возникает в результате появления предупреждений от системы запуска автоматизированного тестирования, вызванных определенными работами специалистов по АТ в программе;

– issue with design change – данный баг появляется в тех случаях, когда внесенные в спецификации и дизайн изменения не предусмотрены заранее перед запуском автоматизированных тест-кейсов.

При нахождении дефекта в программе специалист по обеспечению качества передает информацию с определенной детализацией специалисту по разработке для дальнейшего устранения проблемы [21]. К данной информации относятся следующие атрибуты:

– номер отслеживаемого дефекта (Bug number) – необходим для быстрой коммуникации между специалистами и для дальнейшем отчетности по выявленным багам проекта;

– краткое описание бага (Summary) – сжатое информативное описание основы бага;

– описание и шаги для воспроизведения дефекта (description and steps to reproduce) – детальное описание найденного бага, пройденные шаги специалиста по обеспечению качества до момента появления дефекта, дополнительные комментарии при необходимости, постановка ожидаемого и фактического результата на упавшем шаге;

– приложение (attachment) – предоставление наглядных результатов фактического результата при дефекте, а также ссылки на необходимые спецификации и дизайны, описывающие требования к ожидаемому результату;

– автор найденного бага (Submitted by) – указание специалиста, нашедшего дефект, для дальнейшей коммуникации с ним и для уточнения не указанных выше деталей бага;

– дата и время появления бага (Date submitted) – указывается момент создания бага в системе отслеживания ошибок проекта, который позволяет специалистам по разработке находить необходимые файлы и уведомления, созданные в базе данных на момент воспроизведения дефекта в программном обеспечении;

– ответственный за баг (Assigned to) – специалист, который должен решить найденную проблему;

– передавший баг специалист (Assigned by) – указывается специалист, который передал данный баг ответственному, то есть тот, кто заполнял параметр Assigned to;

– компонент (Component) – определенная функциональная часть программного обеспечения (модуль), к которой относится найденный дефект или с которой связан;

– версия, в которой найден дефект (version found);

– сборка с уникальным номером, в которой найден дефект (Build found);

– приоритет найденного бага (Priority) – указывается коммерческая важность дефекта с точки зрения бизнес-требований, которая может повлиять на процент затронутых пользователей, принести денежные потери для бизнеса, приобрести негативные юридические последствия или последствия для имиджа компании заказчика;

– серьезность бага (Severity) – указывает на степень воздействия технического аспекта бага на программное обеспечение в виде абсолютной категории, к которым можно отнести системные сбои программы, зависание модулей, нарушение безопасности данных;

– список оповещаемых специалистов (Notify list / Watchers) – список участников проекта, которые несут ответственность за найденный дефект или нуждаются в постоянном оповещении об изменениях данного дефекта (например, руководитель отдела обеспечения качества должен быть уведомлен о каждом статусе дефекта).

После получения детальной информации, приведенной выше, специалист отдела разработки находит начальную причину выявленного бага (root cause) и приступает к поиску решения проблемы. Данные шаги являются элементами жизненного цикла найденных багов, которые отслеживаются при помощи статуса [17].

Статус бага (status) – позволяет фиксировать этап жизненного цикла дефекта. На момент нахождения бага статус выставляется «Открыт» (open), в период разработки исправления и поиска решения – «В процессе» (fix/build in progress), после решения проблемы специалист по разработке выставляет статус «Решено» (fixed) и дефект передается специалисту по обеспечению качества и переходит в статус «готов к проверке» (ready for verify/ready for testing). На практике встречаются ситуации, при которых специалист отдела разработки не может воспроизвести указанный баг, поэтому вынужден указать статус дефекта «не воспроизводится» (can’t reproduce) и вернуть нашедшему для повторного воспроизведения. В результате проведения детального анализа специалист по разработке может сделать вывод о том, что найденная проблема не является дефектом и не требует исправлений, так как выявленный случай не противоречит спецификации или бизнес-требованиям заказчика и не блокирует дальнейшие действия и функции проекта. Статус найденной проблемы становится «Не баг» (not a bug). Соответственно специалист отдела разработки должен предоставить ссылки на необходимые отрывки

документации, спецификации или дизайна, а также выводы по консультации с бизнес-аналитиками и руководителями основным отделов проекта.

При устранении реального дефекта создается следующий билд с новым уникальным номером, который далее указывается в информации по дефекту и передается специалисту по обеспечению качества для проверки. Таким образом, к атрибутам найденного бага дополняются следующие:

- версия программы с устраненной проблемой (fixed version),
- версия билда с починенным кодом (fixed build),
- причина появления бага (root cause),
- комментарии специалиста по разработке (comments) указываются при необходимости.

Специалист по обеспечению качества устанавливает новую сборку и проводит повторную проверку, воспроизводя шаги до момента появления дефекта. В случае, если найденный ранее баг не устранен, устанавливается статус «Безуспешно» (verification failed) и дефект снова передается специалисту по разработке с дополнительными комментариями. В ином случае, когда баг не повторяется и фактический результат соответствует ожидаемому строго по спецификации, статус устанавливается «Успешно» (fix is verified/test passed). По результатам проведения анализа решенной проблемы формируются необходимые выводы для избегания повторной ошибки, после чего дефект закрывает со статусом «Закрыт» (closed).

Таким образом, основная задача каждого из участников проекта выявить и устранить возможные блокирующие и серьезные дефекты программного обеспечения, а также по возможности максимально избежать последствий и повторных появлений проблем. Существуют различные виды дефектов как со стороны самого решения и имплементации, так и сторонние дефекты тестовой среды, интеграции компонентов, инструментов автоматизированного тестирования или даже в технической документации. Каждый найденный баг обладает жизненным циклом в процессе устранения и должен быть описан максимально детально. Баги в программном обеспечении могут принести

значительные негативные результаты и со стороны технических аспектов, и со стороны финансовой и бизнес-стратегии компании заказчика проекта.

2.3 Принципы и алгоритм тестирования новых функциональностей и выполнения регрессивного тестирования

Регрессивное тестирование программного обеспечения представляет собой повторную проверку уже подготовленного релиза после каждого создания новой версии, предназначенную для подтверждения того, что внесенные изменения и добавления в код не повлияли на ту часть проекта, в которой не изменялся код.

Регрессивное тестирование рекомендуется проводить каждый раз после корректировки программы или сайта, которая может включать исправление дефектов, слияние кода, миграцию на другую операционную систему или базу данных, добавление новой функциональности и другие изменения. Если в процессе эксплуатации программного обеспечения существенно выросло число пользователей системы, рекомендуется проводить регрессивное нагрузочное тестирование.

С повышением сложности системы и увеличением количества функций увеличивается и количество тестов, а значит, и время на их повторение. Поэтому регрессивное тестирование далеко не всегда означает полную проверку всей функциональности. Чтобы гарантировать высокое качество выпускаемой версии программы при сохранении оптимальных сроков и стоимости, необходим продуманный подход для определения стратегии и достаточного регрессивного покрытия. В ходе проекта специалисты отдела обеспечения качества, в первую очередь, проверяют ту часть функциональных модулей, где вероятность появления ошибки после внесенных изменений наиболее велика [18].

Если проект подвергается частым изменениям и доработкам, то потребность в одинаковых проверках может значительно возрасти. Для

экономии времени специалисты отдела обеспечения качества формируют автоматизированные регрессивные тестовые сценарии, благодаря которым сокращаются сроки тестирования без потери в качестве работ.

Для нахождения дефектов при проведении автоматизированного тестирования необходимо выполнять детальный анализ каждого тест-кейса. Когда процесс автоматизированного тестирования завершается, в сторонней программе, самостоятельно прошедшей тест-кейсы, отображается в большинстве случаев древовидный отчет по пройденным кейсам, статусы которых могут быть следующие: failed (провален, остановился на упавшем шаге; обычно окрашен в красный цвет), warning (предупреждение, приемлемое для полного завершения автотеста; желтый цвет), terminated (прекращен из-за сбоя АТ системы или остановлен специалистом; оранжевый цвет), passed (пройден успешно, считается «счастьем» для аналитика результатов АТ; зеленый цвет).

Анализ результатов автоматизированного тестирования начинается, как правило, с кейсов в статусе failed. Иногда причина дефекта отображается сразу в информации по упавшему шагу в АТ системе, однако бывают случаи, когда очевидная причина отсутствует и требуется более детальная проверка.

Следующим приемлемым действием аналитика АТ является проверка такого же кейса вручную в тестовой среде. В средних или масштабных проектах, при условии успешной разработки программного обеспечения, предусмотрены валидационные[11] сообщения о текущих актуальных ошибках сервера, системы, сети или самих шагов в кейсе в момент выполнения действия в тестовой среде. Примерами таких валидаций могут выступать уведомления о перегрузке системы большим количеством действий или о незаполненных обязательных параметрах, необходимых для продолжения кейса.

[11] Валидация — это проверка систем, процессов, пользователей на то, насколько они соответствует определенным установленным требованиям.

В таких случаях, когда валидации при воспроизведении вручную бага, выявленного в автоматизированном тест-кейсе, не позволяют определить причину данного дефекта, информация передается отделу разработки для анализа, выявления совокупности ошибочного кода и дальнейшего исправления по алгоритму, описанному в предыдущем пункте данной работы.

Во время анализа тест-кейсов со статусом warning следует изучить причину появления данных предупреждений и в случаях их постоянного возникновения необходимо собрать полученные выводы и передать отделу разработки. Например, подобные уведомления могут появляться по причине долговременной загрузки одной страницы сайта, которая должна была появиться значительно быстрее. Данную проблему следует осветить специалисту по разработке, ответственному за контроль скорости работы программного обеспечения.

Автоматизированные тест-кейсы обычно получают статус terminated, когда специалист по анализу АТ целенаправленно останавливает тест-кейсы (например, во время формирования нового автоматизированного тестового сценария пробная попытка была безуспешна и дальнейшее его продолжение не нужно), или, когда специалисты по обеспечению работоспособности системы для автоматизированного тестирования вынуждены установить новую версию или исправить ошибки данной системы.

Тестовые сценарии, получившие статус passed, показывают положительный результат регрессивного тестирования, при котором новые внесенные изменения и добавления в код не заблокировали и не сломали существующий код. Данные выводы позволяют сформировать необходимый отчет и передать готовую версию в релиз.

Если результатом автоматизированного тест-кейса со статусом failed является баг, не связанный с ошибкой написанного автотеста или с конфигурацией тестового окружения, то создается экземпляр со всеми перечисленными в предыдущем пункте данной работы атрибутами в системе отслеживания дефектов (bug tracking system). Принцип работы данной

системы представлен на рисунке 4. Участниками процесса разработки проекта в системе отслеживания ошибок могут быть не только специалисты отделов разработки и обеспечения качества, но и заказчик проекта, и пользователи посредством передачи своих пожеланий и жалоб через заказчика.

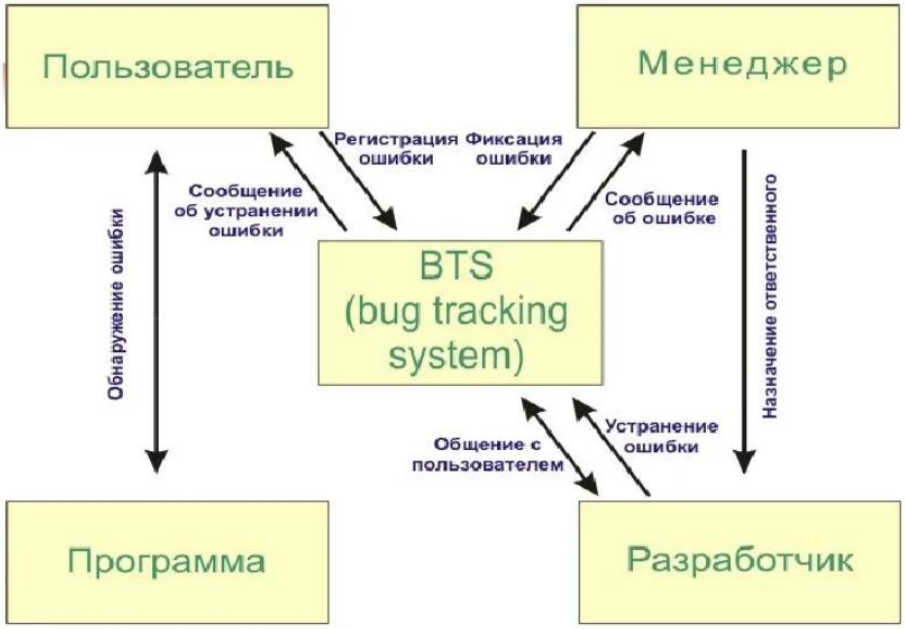


Рисунок 4 – Принцип работы bug tracking system

Примерами таких систем отслеживания ошибок выступают Youtrack, Bugzilla, Redmine, JIRA, Fogbugz, Clearquest, Trackstudio, Bontq и другие. Данные программы предназначены для учета и контроля ошибок, дефектов и неполадок, обнаруженных в программном обеспечении, а также для создания и отслеживания задач и пожеланий от заказчика проекта. Некоторые системы отслеживания ошибок обладают широким спектром функций и преимуществ, а именно позволяют эффективно автоматизировать рабочий процесс, упростить процесс распределения обязанностей между участниками проекта, быстро формировать необходимые отчеты и строить графики рабочего процесса [17].

При регулярной поддержке среднего и крупного проекта появляется необходимость в постоянном исправлении кода, устранении дефектов,

улучшении программы, а также в изменении функциональности по требованию и пожеланию заказчика проекта.

Внедрение новой функциональности начинается по классической методологии разработки программного обеспечения, а именно с формирования бизнес-требований, технических и функциональных требований к новой функциональности. Следующим этапом необходимы разработка стратегии тестирования и составление тестового плана.

С целью определения наиболее оптимального количества тест-кейсов применима формула цикломатической сложности Маккейба для максимального покрытия всех возможных решений [26]. Представим, что новая функциональность представлена в виде следующего отрывка кода:

```
{code}
module nonsense()
/* a[] and b[] are global variables */
begin
int i,x
i = 1
read (x)
while (i < x) do begin
a[i] = b[i] * x
if a[i] > 50 then
print (‘‘array a is over the limit’’)
else
print (‘‘ok’’)
i = i +1
end
print (‘‘end of nonsense’’)
end
{code}
```

Рассмотрим простой пример вычисления количества тестовых сценариев на основе блок-схемы по представленному выше коду, изображенной на рисунке 5.

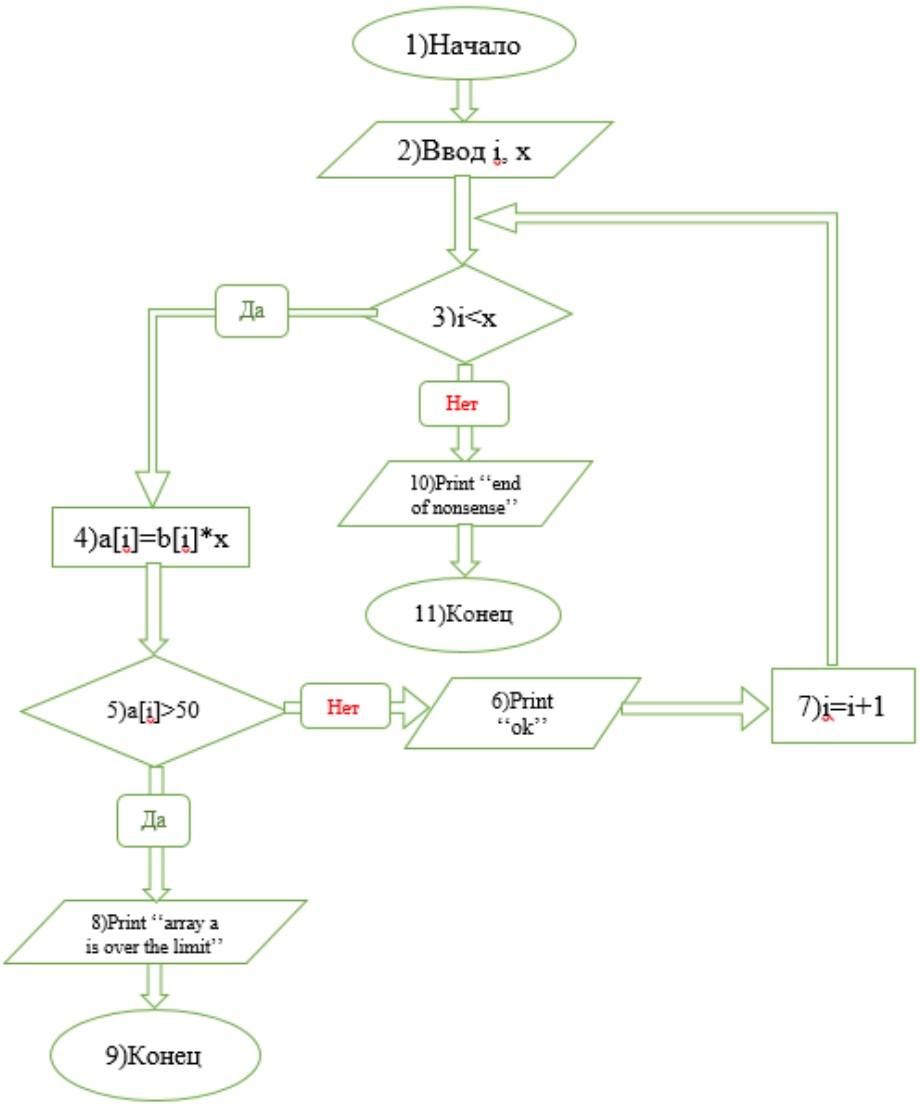


Рисунок 5 – Блок-схема исходного кода

Cyclomatic complexity value (McCabe's Cyclomatic Complexity) – это результат вычисления формулы V(G), где E (edge) – количество ребер (связей двух соседних узлов блок-схемы), N (node) – количество узлов (вершин блок-схемы).

Цикломатическая сложность Маккейба вычисляется на основе вышеуказанного примера блок-схемы по формуле (1):

$$V(G) = E - N + 2. \quad (1)$$

На изображенной блок-схеме одиннадцать ребер и одиннадцать узлов, рассчитаем цикломатическую сложность: $V(G) = 11 - 11 + 2 = 2$.

Независимый путь представляет собой путь, имеющий хотя бы одно ребро, которое ранее не использовалось ни в одном другом пути.

Линейно-независимые пути по блок-схеме будут следующие:

1) 1-2-3-4-5-6-7-3-10-11

2) 1-2-3-4-5-8-9

В результате количество независимых путей всего два – это цикломатическая сложность исходного кода по блок-схеме. Соответственно количество тест-кейсов для достижения максимального покрытия решений в данном случае – 2. По результату вычисления цикломатической сложности Маккейба формируются тест-кейсы и вносятся в план тестирования.

В плане тестирования должно быть приведено описание критериев возможных дефектов, найденных в процессе дальнейшего тестирования, и критериев доказательств об их устранении [16]. Приблизительный пример представлен в виде таблицы 1.

Таблица 1 – Критерии оценки дефектов и их устранения

Название	**Критерий**
Критический / высокий дефект, блокирующий процесс тестирования программы	Если дефект после нахождения вынуждает останавливать тестирование, влияет на процесс дальнейшего выполнения тест-кейсов
Проблемы программной среды	Если дефект выявлен при нестабильной работе программной среды и если дефект привел к остановке тестирования
Исправление / устранение дефекта	Отчеты, доказывающие разрешение / устранение дефекта
Устранение проблем программной среды	Отчеты, доказывающие разрешение проблем программной среды

Процесс подготовки к тестированию и непосредственный процесс тестирования реализуется с использованием системы отслеживания ошибок. Все составленные тестовые сценарии вносятся в систему и распределяется между специалистами по обеспечению качества. Руководитель отдела обеспечения качества регулирует процесс тестирования и руководит процессом устранения найденных дефектов [1].

На каждом этапе тестирования команда должна отчитываться о результатах проведения тест-кейсов и исправления дефектов.

Отчет о тестировании содержит данные о состоянии дефекта, о состоянии выполнения тест-кейсов и следующую информацию:

– список дефектов, готовых к тестированию, с результатом тестирования: «пройден», «не пройден» или «заблокирован» (не может быть протестирован);

– список тест-кейсов с результатами выполнения: Pass, Pass with minor defects (с незначительными дефектами), Fail или Blocked. Тест-кейсы содержат ссылку на соответствующий дефект из списка найденных дефектов.

В тест-кейсах со статусом Failed указаны ссылки на найденные дефекты, которые были оценены как важные баги, влекущие за собой негативные последствия. Статус Failed указан, если найден значительный дефект и фактический результат не совпал с ожидаемым по требованиям.

Статус Passed указан, если все фактические результаты совпали с ожидаемыми, не обнаружено критических дефектов, влияющих на работоспособность новой функциональности.

Статус Blocked указан у тест-кейсов, которые не могут быть выполнены из-за найденных дефектов.

Статус тест-кейса Pass with minor defects указан в случае наличия незначительных (например, косметических) дефектов, не влияющих на работу функциональности.

После завершения этапа тестирования новой функциональности необходимо провести регрессивное тестирование для того, чтобы убедиться в

исправной работоспособности всего проекта, включая незатронутую изменениями часть [29].

Успешный результат обеспечения качества новой функциональности оценивается следующими путями [7]:

– готовое решение соответствует функциональным и нефункциональным требованиям в полном объеме;

– дефекты, не соответствующие требованиям, исправлены;

– валидация готового решения пройдена в соответствии со спецификациями, бизнес-требованиями и дизайнами;

– новые функциональности программы работают исправно;

– все тестовые исключения охвачены и выполнены.

Таким образом, регрессивное тестирование позволяет своевременно выявить и устранить дефекты программного обеспечения до выдачи версии в релиз пользователям. Данное тестирование может проводиться как при помощи инструментов автоматизации, так и вручную специалистами по обеспечению качества, метод оценки качества программного обеспечения выбирается в зависимости от ситуации. Составление плана тестирования при создании новой функциональности в программном обеспечении выполняет важную функцию для регулирования дальнейшего тестирования. Грамотное планирование позволит своевременно выявить дефекты разработанного модуля и избежать трудностей при завершении приемочного тестирования. Эффективным инструментом управления проектом является система отслеживания ошибок, которая способствует своевременному реагированию на найденные ошибки и их быстрому устранению.

Глава 3 Реализация оценки качества релиза на примере приложения «Menu 1.0»

3.1 Формирование технической документации приложения «Menu 1.0»

Процесс формирования технической документации для приложения «Menu 1.0» и планирование тестирования основаны на теоретических знаниях и практических навыках, приобретенных во время прохождения производственной практики в компании ООО «НетКрэкер» в России.

Netcracker Technology corporation – это динамически развивающаяся международная компания в области информационных технологий и телекоммуникаций, основанная в 1993 году и предоставляющая услуги по разработке и усовершенствованию облачных и цифровых решений для ведения бизнеса [30].

Компания ООО «НетКрэкер», основанная как дочерняя компания Netcracker Technology corporation, занимает лидирующие позиции на рынке в области разработки и поддержки информационных технологий в России и, в частности, по Самарской области. Деятельность организации ООО «НетКрэкер» заключается в планировании, обследовании, разработке, управлении и сопровождении компьютерного программного обеспечения, а также в консультации в области компьютерных технологий. Штат сотрудников в компании состоит из квалифицированных специалистов, получивший свой опыт в ходе непосредственной работы над проектами компании.

Заказчиком проекта по реализации приложения «Menu 1.0» является предприятие общественного питания, бизнес-целью которой является автоматизация организационной деятельности путем ускорения процесса получения заказов от клиентов и их оплаты. Рассмотрим дерево целей предприятия, изображенного на рисунке 6.

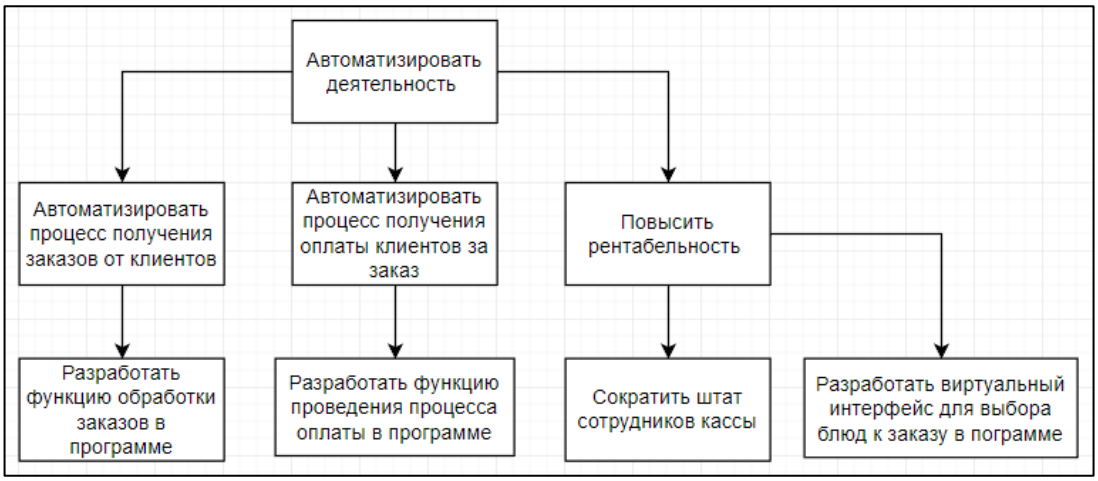


Рисунок 6 – Дерево целей предприятия общественного питания

С точки зрения предприятия-заказчика процесс реализации проекта, изображенного на рисунке 7, осуществляется шестью этапами, включая взаимодействие с IT-компанией.

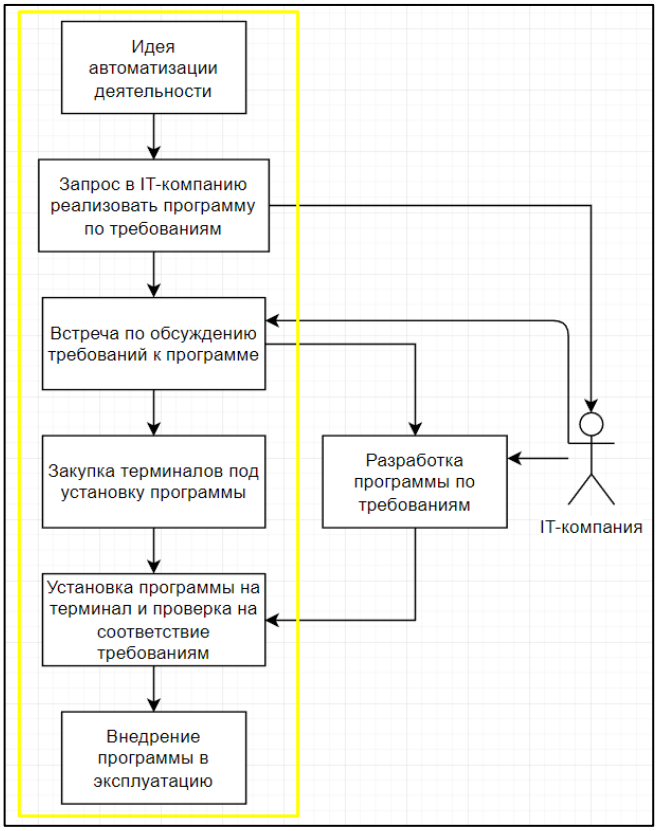


Рисунок 7 – Дерево целей предприятия общественного питания

Целевая аудитория предполагаемого приложения «Menu 1.0» состоит из людей разных возрастов. Привычки посетителей мест общественного питания гарантируют сохранение высокого спроса на кафе и рестораны в течение следующих пяти лет. Быстрый рост сети Internet и информационных технологий, используемых во всем мире, стимулирует появление экономики, основанной на инфраструктуре современных технологий. С учетом острой конкуренции владельцы предприятий общественного питания должны стремиться выделиться среди остальных соперников.

Автоматизирующее приложение «Menu 1.0» позволит предприятию общественного питания открыть такие возможности, как ускорение процесса получения заказов от посетителей, автоматизация расчета итоговой суммы оплаты за заказ и исключение ошибок на основе человеческого фактора, планирование расширения бизнеса. Внедрение автоматизированного процесса оформления заказов также позволит сохранить бизнес в случае повторного ужесточения правил безопасности людей в период пандемии в общественных местах.

Итак, для сбора технический требований к приложению «Menu 1.0», в первую очередь, необходимо выявить назначение программы, которое необходимо будет удовлетворить в результате реализации проекта.

Основной целью внедрения данного мобильного приложения является сбор заказов клиентов кафе и формирование итогового чека с учётом акций организации. Приложение «Menu 1.0» позволит клиентам кафе выбрать тип меню (завтрак, обед, ужин) и желаемые товары (напиток, первое блюдо, выпечка и тому подобное), добавить их в корзину, оформить заказ и получить чек по электронной почте.

На первый взгляд, указанной информации может быть достаточно для перехода к этапу написания кода приложения «Menu 1.0», однако стоит разобрать каждый бизнес-процесс более детально и продумать каждое действие так, как должна быть построена логика программы, с целью формирования качественной технической документации проекта.

За появлением дефектов стоит следить с первого этапа разработки приложения, так как, в первую очередь, ненадежным является человеческий фактор при формировании технической документации. За своевременный контроль и устранение дефектов несет ответственность каждый участник проекта. Устранение ошибок и дефектов на ранних этапах планирования проекта позволит значительно сэкономить финансовые и временные ресурсы [32].

Рассмотрим этапы разработки приложения «Menu 1.0» по каскадной методологии разработки проекта, изображенного на рисунке 8, с визуализацией каждого этапа. Важным этапом обеспечения качества приложения «Menu 1.0» является анализ и формирование дизайна, в процессе которого уже должна быть подготовлена стратегия и план тестирования, а также тестовые сценарии, покрывающие все возможные решения в приложении. На момент реализации тестирования команда по обеспечению качества должна провести проверку подготовленной версии приложения на предмет наличия дефектов.

Процесс работы приложения «Menu 1.0» нужно декомпозировать на отдельные бизнес-процессы с целью обнаружения скрытых сценариев и уточнения дополнительных требований к поведению приложения:

– поиск товаров по типу меню;

– добавление товара определенного типа меню в корзину;

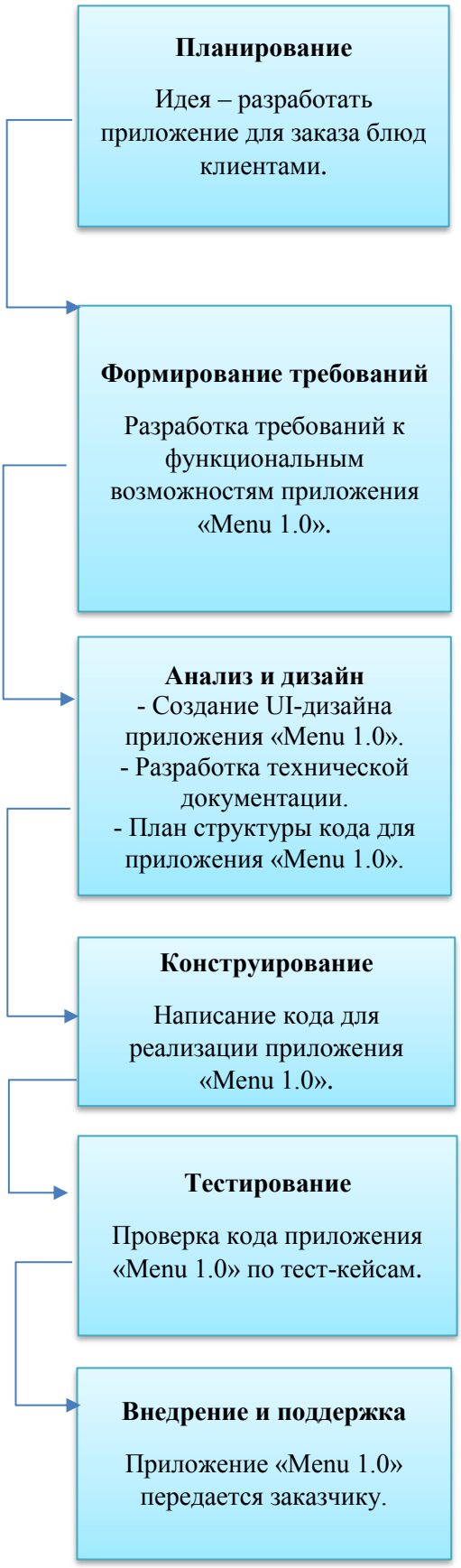
– выбор оплаты заказа (по банковской карте онлайн, виртуальным счетом, наличными средствами при получении заказа);

– отправка списка товаров в виде заказа организации для приема заказа;

– получение чека по электронной почте пользователем;

– ознакомление с информацией о приложении «Menu 1.0»;

– ознакомление с инструкцией по работе с приложением «Menu1.0».



Стадия Планирования.
Появилась идея разработать приложение для заказа блюд клиентами, при помощи которого можно будет сохранять список блюд в актуальном состоянии и принимать заказы от клиентов. На данной стадии можно поставить срок 1 месяц, выбрать язык программирования (например, код будет написан на Java) или специальную программу, сформировать участников разработки.

Стадия Формирования требований.
- В приложении «Menu 1.0» должна быть возможность добавлять и удалять товары из корзины.
- Принимать заказы и отправлять чеки клиентам на электронную почту.
- Использовать расчет актуальных скидок на определенные товары и категории.
- На каких устройствах можно будет использовать приложение.

Стадия Анализа и дизайна.
- Создается дизайн User Interface проекта (где какие кнопки будут располагаться).
- Программист изучает спецификацию и планирует каким образом будет строить будущий код приложения «Menu 1.0» и верстать его.
- QA-специалист изучает спецификацию и дизайн, а также разрабатывает тест-кейсы и тестовую документацию.

Стадия Конструирования.
Разработчик пишет код и выполняет юнит-тестирование (проверяет наиболее приоритетные сценарии, например, правильно ли выполняет расчет скидки).

Стадия Интеграции и тестирования.
- Если бы код был громоздкий и его писали бы несколько программистов, то была бы необходима интеграция модулей приложения.
- Проводится тестирование по заранее созданным тест-кейсам и устраняются дефекты.

Стадия Внедрения и эксплуатации.
Приложение «Menu 1.0» прошло приемочное тестирование и можно запускать его в релиз для клиентов заказчика.

Рисунок 8 – Каскадная методология разработки приложения «Menu 1.0»

Разберем бизнес-процесс «получение чека по электронной почте» и представим его в виде схемы цикла мышления программы на доступном языке для заинтересованных участников проекта, не обладающих знаниями программиста.

Рассмотрим наиболее благоприятный сценарий работы пользователя в приложении:

– пользователь нажимает кнопку [Отправить чек на почту];

– пользователь вводит корректную электронную почту, используя раскладку клавиатуры на английском языке, нижний регистр и знак «@»;

– пользователь нажимает кнопку [ОК], программа отправляет чек;

– пользователь получает чек на электронную почту.

Представим негативные сценарии работы пользователя, которые также должно учитывать приложение и соответствующе реагировать:

– пользователь нажимает кнопку [Отправить чек на почту];

– пользователь вводит электронную почту.

Пользователь вводит электронную почту:

– не используя строку заполнения почты (пустое значение);

– используя раскладку клавиатуры на русском языке;

– используя верхний регистр (или разный регистр, например, TeSt@EmAiL.rU);

– не используя знак «@», указав только адрес почты до данного знака;

– используя пробелы, математические и пунктуационные символы (помимо точки).

В результате анализа приведенных выше сценариев необходимо разработать план реакции приложения «Menu 1.0» на каждый случай. Действия программы должны учитывать целевую аудиторию, поэтому необходимо добавлять к уведомлениям об ошибках дополнительную информацию о причине ошибки и способе её избегания. Наглядный цикл реакций приложения на негативные сценарии представлены на рисунке 9.

В соответствии с планом реализации приложения «Menu 1.0» в бизнес-процессе «получение чека по электронной почте» все уведомления об ошибках должны собираться в массив и в результате прохождения цикла выводить массив на экран пользователю. При прохождении позитивного сценария пользователем после нажатия кнопки [Отправить чек на почту] программа выполняет весь цикл проверки почты перед фактической отправкой чека.

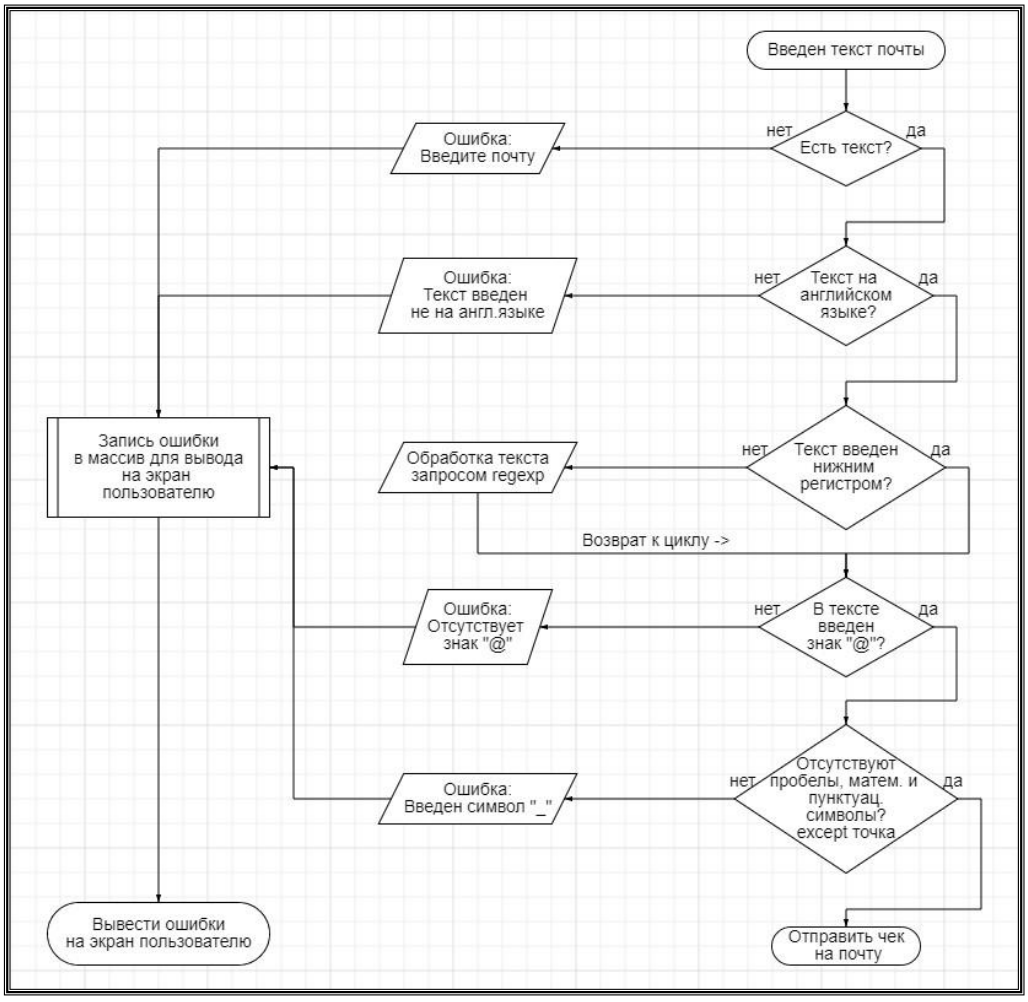


Рисунок 9 – Цикл логики приложения «Menu 1.0» по бизнес-процессу «получение чека по электронной почте»

Представим взаимодействие пользователя с программой в бизнес-процессе «получение чека по электронной почте» в виде UML-диаграммы на рисунке 10.

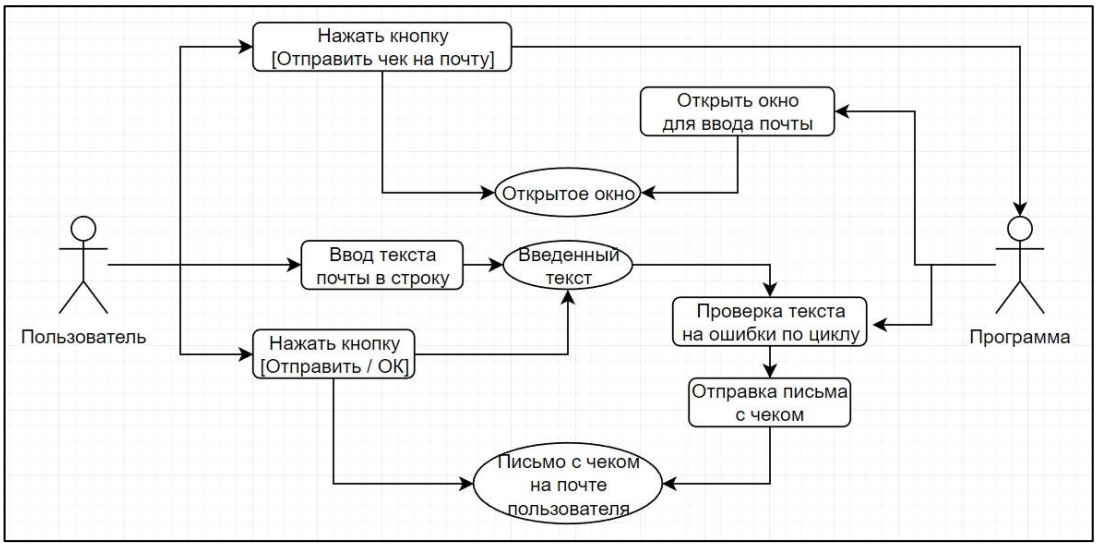


Рисунок 10 – UML-диаграмма приложения «Menu 1.0» по бизнес-процессу «получение чека по электронной почте»

Проанализировав UML-диаграмму, можно сделать вывод о том, что пользователю доступно три действия в приложении для выполнения данного бизнес-процесса, на каждое из которых должно быть предусмотрено определенное поведение приложения, основанное на циклической логике.

Учитывая вышеуказанные мероприятия, необходимо разрабатывать стратегию поведения программы при каждом взаимодействии пользователя с ней для всех бизнес-процессов.

Для дополнительного улучшения работы программы следует добавить функцию поиска товара по вводимому тексту пользователем в поисковом поле с целью упрощения задачи для пользователя. Данная функцию позволит организации-заказчику приобрести клиентоориентированность, а также привлечь и сохранить своих клиентов.

Внедрение новой функциональности будет реализовано на том этапе разработки проекта, когда все основные бизнес-процессы будут готовы к работе для пользователя и этап тестирования данных процессов будет завершен успешно.

Таким образом, после постановки необходимых требований к каждому действию в бизнес-процессах важно утвердить сформированную

спецификацию с заказчиком, программистами и специалистами по тестированию проекта. Декомпозиция бизнес-процессов позволит выявить скрытые сценарии и предусмотреть дополнительные требования к логике программы. При подтверждении всех требований к проекту можно переходить к следующему этапу разработки приложения «Menu 1.0». Для написания кода команда разработки должна основываться на технической документации и нацеливать свою работу на тот ожидаемый результат, который был утвержден заказчиком проекта и командой бизнес-аналитиков. В свою очередь, команда по тестированию и обеспечению качества приложения «Menu 1.0» должна разработать план и стратегию тестирования с целью проверить приложение на наличие дефектов и несоответствий с логикой, бизнес-целями, ожиданиями пользователей и заказчика.

3.2 Планирование этапа тестирования приложения «Menu 1.0»

Подготовка к этапу тестирования приложения должна инициироваться на ранних стадиях разработки проекта, а именно при анализе и формировании технической документации проекта.

В планировании тестирования приложения «Menu 1.0» необходимо, в первую очередь, подготовить и организовать команду по обеспечению качества. Рассмотрим роли и обязанности участников команды тестирования приложения «Menu 1.0».

Руководитель команды обеспечения качества (QA Lead) отвечает за выполнение следующих задач:

- подготовка тестовой стратегии, плана тестирования и графиков;
- участие в создании и изменении плана проекта;
- контроль за сроками выполнения тестирования;
- обеспечение качества и согласованности результатов тестирования;

– оповещение руководителя проекта о потенциальных и существующих проблемах проекта;

– подготовка и сопровождение всей технической документации тестирования;

– коммуникация с бизнес-аналитиками, непосредственным руководством и командой тестирования по вопросам возникших проблем тестирования.

Специалист в области тестирования (QA engineer) отвечает за выполнение задач:

– просмотр доступных тестовых сценариев;

– подготовка и оценка спецификации тестирования;

– подготовка предварительных тестовых данных;

– выполнение тестовых сценариев;

– сообщение о дефектах программы руководителю и команде разработки;

– коммуникация с разработчиками, руководителем проекта, командой тестирования;

– отчет о выполнении ежедневных задач от руководителя;

– подготовка данных для автоматизированного тестирования.

Аналитик этапа тестирования (QA Analyst) отвечает за выполнение следующих задач:

– анализ требований, проектной технической документации, требований тестирования;

– коммуникация с командой тестирования по вопросам спецификации для реализации тестирования;

– анализ технической документации тестирования с точки зрения бизнеса;

– получение доступа к процессу тестирования;

– оповещение руководителя проекта о проблемах тестирования;

– консультация и координация команды тестирования по вопросам бизнес-процессов в проекте.

Предварительные требования для запуска процесса тестирования приложения «Menu 1.0» представлены в виде таблицы 2.

Таблица 2 – Подготовка к запуску тестирования

Шаг	**Описание**
1	Получение бизнес-требований, функциональных и нефункциональных требований к программе «Menu 1.0»
2	Создание плана проекта и плана ресурсов
3	Создание стратегии тестирования
4	Создание плана тестирования
5	Создание тестовых сценариев
6	Согласование порядка приемки программы с заказчиком

Тестирование приложения «Menu 1.0» делится на определенные этапы реализации в зависимости от метода тестирования. Для каждого этапа описывается назначение к области приложения, готовятся предварительные необходимые условия для инициации этапа, заносится ожидаемый результат реализации в соответствии со спецификацией приложения «Menu 1.0» и назначается ответственный участник команды тестирования. Распишем наиболее приближенный к реальным вариант оформления этапов реализации тестирования приложения «Menu 1.0» для технической документации по обеспечению качества в виде таблиц 3-9.

Первым этапом тестирования должны проверяться основные функции и задачи приложения «Menu 1.0». За процесс функционального тестирования должны быть назначены ответственные участники команды по обеспечению качества, а по результатам реализации ответственные должны предоставить отчет по тестированию с указанием статусов прохождения.

Таблица 3 – Этап функционального тестирования

Этап тестирования	Описание	Ответственность
Functional testing	Исправность работы функций программы (добавление, удаление блюд в заказе, подтверждение и отмена заказа, отправка чека на почту, подсчет акций в чеке).	QA Manager, QA engineer
Входные данные	Программа запущена, нет критических проблем с функционалом для выполнения тестирования.	
Выходные данные	Все тест-кейсы выполняются по требованиям, все дефекты задокументированы, нет нерешенных приоритетных дефектов по проверке функций программы.	
Результаты	Сводный отчет по тестированию приложения	

Второй этап тестирования заключается в прохождении более детальных ожидаемых сценариев работы приложения «Menu 1.0» с использованием метода позитивного тестирования.

Таблица 4 – Этап позитивного тестирования

Этап тестирования	Описание	Ответственность
Positive testing	Выполнение позитивных сценариев использования программы, которые предполагают нормальную / правильную работу системы.	QA Manager, QA engineer
Входные данные	Программа запущена, нет критических проблем в программе перед выполнением тестирования.	
Выходные данные	Все тест-кейсы выполняются по требованиям, все дефекты задокументированы, нет нерешенных приоритетных дефектов.	
Результаты	Сводный отчет по тестированию приложения	

Негативный метод тестирования выполняется следующим этапом с целью проверки потенциальных ошибок во взаимодействии пользователя с приложением «Menu 1.0» и на поиск возможных дефектов в логике кода.

Таблица 5 – Этап негативного тестирования

Этап тестирования	Описание	Ответственность
Negative testing	Использование программы с потенциально возможными ошибками пользователей или ошибками системы.	QA Manager, QA engineer
Входные данные	Программа запущена, нет критических проблем в программе перед выполнением тестирования.	
Выходные данные	Все тест-кейсы выполняются по требованиям, все дефекты задокументированы, нет нерешенных приоритетных дефектов.	
Результаты	Сводный отчет по тестированию приложения	

Предусмотрев возможную нагрузку приложения «Menu 1.0» большим количеством пользователей команда по обеспечению качества должна проверить скорость и надежность работы приложения. Для реализации данного метода тестирования команда использует сторонние инструменты или программы, имитирующие наплыв пользователей в приложение и активное использование его функций.

Таблица 6 – Этап тестирования скорости и надежности

Этап тестирования	Описание	Ответственность
Performance testing	Проверка скорости работы программы и возможностей нагрузки большим количеством пользователей.	QA Manager, QA engineer
Входные данные	Программа запущена, нет критических проблем в программе перед выполнением тестирования.	
Выходные данные	Программа своевременно выполняет заложенные функции и не зависает. Программа работает исправно при большой нагрузке пользователями. Все тест-кейсы выполняются, все дефекты задокументированы, нет нерешенных приоритетных дефектов.	
Результаты	Сводный отчет по тестированию приложения	

Следующим этапом тестирования приложения «Menu 1.0» должна быть предусмотрена проверка поведения приложения в случае многократной

активации функций, превышающей ожидаемое взаимодействие пользователя с приложением.

Таблица 7 – Этап стресс-тестирования

Этап тестирования	Описание	Ответственность
Stress testing	Проверка реакции программы на превышение предела нормального, ожидаемого функционирования.	QA Manager, QA engineer
Входные данные	Программа запущена, нет критических проблем в программе перед выполнением тестирования.	
Выходные данные	Реакция программы на превышение пределов сценария пользователя продумана и работает исправно. Все тест-кейсы выполняются, все дефекты задокументированы, нет нерешенных приоритетных дефектов.	
Результаты	Сводный отчет по тестированию приложения	

Важным этапом тестирования перед выдачей релиза в эксплуатацию является проверка пользовательского интерфейса на предмет удобства и интуитивного использования приложения «Menu 1.0» с позиции нового пользователя.

Таблица 8 – Этап тестирования удобства пользователя

Этап тестирования	Описание	Ответственность
Usability testing	Оценка удобства пользования клиентом интерфейса и функций программы.	QA Manager, QA engineer
Входные данные	Программа запущена, нет критических проблем в программе перед выполнением тестирования.	
Выходные данные	Все тест-кейсы выполняются по требованиям, все дефекты задокументированы, нет нерешенных приоритетных дефектов.	
Результаты	Сводный отчет по тестированию приложения	

Завершающим этапом тестирования проводится финальная проверка приложения «Menu 1.0», установленного на внешнем окружении заказчика.

Приемочное тестирование необходимо для утверждения конечной версии релиза приложения, подготовленного после устранения всех критичных и блокирующих дефектов командой разработки кода (см. Приложение А, Таблица А.1). На данном этапе проводится подготовка и загрузка предварительных тестовых данных на основе реальных, таких как актуальное меню кафе, цены на каждое блюдо, скидки и их ограничения.

Приемочное тестирование должно проводится только на вышеуказанных данных и охватывать важные функциональные и нефункциональные тестовые сценарии, демонстрирующие заказчику качество реализованного приложения «Menu 1.0» в соответствии с установленными в начале цикла разработки требованиями. Результаты данного метода тестирования должны быть только положительными, или с возможными допущениями дефектов низкого уровня, и должны быть отражены в отчете, который передается по итогу заказчику как отчет о готовом к эксплуатации релизе.

Таблица 9 – Этап приемочного тестирования

Этап тестирования	**Описание**	**Ответственность**
User Acceptance Test (UAT)	Приемочное конечное тестирование для проверки правильности обработки данных и выполнении всех функций программы, заложенных в требованиях.	QA Manager, QA engineer
Входные данные	Программа запущена. Среда для возможности поддержки тестирования подключена. Дефекты тестирования программы рассмотрены и устранены.	
Выходные данные	Все приоритетные тест-кейсы выполнены (ни один не провален), нет нерешенных приоритетных дефектов. Дефекты минимальных приоритетов не мешают принятию программы к эксплуатации. Принято решение о приложении «Accepted».	
Результаты	UAT Test отчет	

Реализация тестирования должна быть ограничена сроками и иногда ресурсами. Например, на весь процесс тестирования выделено десять

специалистов для выполнения сценариев, описанных в плане тестирования. График тестирования по тестовым сценариям, который должен быть подготовлен руководителем команды тестирования и аналитиком, представлен в таблице 10. В большинстве случаев для международной IT-компании вся техническая документация оформляется на английском языке, поэтому реализуем график тестирования в соответствующем виде.

Таблица 10 – График тестирования приложения «Menu 1.0»

Этап тестирования	**TC ID**	**Название сценария**	**Дата выполнения**	**Статус**
Функциональное тестирование	TC01	Добавление блюда клиентом	25.04.2023	Passed
	TC02	Отправить квитанцию на адрес электронной почты	25.04.2023	Passed
	TC03	Открыть информацию о программе	25.04.2023	Passed
	TC04	Удалить блюдо из заказа	25.04.2023	Passed
Позитивное тестирование	TC05	Добавить 3 похожих блюда, нажав на меню	26.04.2023	Passed
	TC06	Отправить квитанцию на верный адрес электронной почты	26.04.2023	Failed
	TC07	Добавление блюд из всех видов блюд (завтрак, обед, ужин)	26.04.2023	Passed
	TC08	Выбрать десерт в соответствии с акцией на ужин	26.04.2023	Failed
	TC09	Закрыть программу с помощью кнопки «выход»	27.04.2023	Failed
	TC10	Выбрать 3 блюда из одного вида меню и полностью удалить их	27.04.2023	Passed
	TC11	Отправить квитанцию на адрес электронной почты перед подтверждением	27.04.2023	Passed
Негативное тестирование	TC12	Отправить квитанцию на неправильный адрес электронной почты	27.04.2023	Failed
	TC13	Отправить квитанцию на пустой адрес электронной почты	28.04.2023	Failed
	TC14	Отправить квитанцию на несколько разных адресов электронной почты	28.04.2023	Failed
	TC15	Пользователь сам выбирает десерт и чай на ужин из меню	28.04.2023	Passed
	TC16	Увеличить количество блюд в строке заказа	29.04.2023	Failed
	TC17	Добавить блюдо без скидки	29.04.2023	Passed
	TC18	Добавить и подтвердить пустой заказ (без блюд)	29.04.2023	Failed

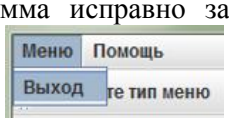
Продолжение Таблицы 10 – График тестирования приложения «Menu 1.0»

Нагрузочное тестирование	ТС19	Заказать все блюда из одного вида меню	04.05.2023	Passed
	ТС20	Добавить 10 похожих блюд и 10 похожих напитков в одном заказе	05.05.2023	Passed
Стресс-тестирование	ТС21	Нажимать все кнопки почти одновременно	02.05.2023	Passed
	ТС22	Создать 5 заказов один за другим	03.05.2023	Passed
Тестирование удобства пользователя	ТС23	Другие пользователи пытаются сделать заказ самостоятельно	30.04.2023	Failed
	ТС24	Другие пользователи пытаются самостоятельно удалить блюда из заказа	30.04.2023	Failed
	ТС25	Другие пользователи пытаются сами отправить квитанцию на адрес электронной почты	01.05.2023	Passed

В процессе тестирования должны проводиться регулярные обсуждения возникших проблем и ограничений тестирования. Результатом будет выступать план действий по смягчению и устранению дефектов и проблем, согласованный с заказчиком и командой проекта.

Для приемочного тестирования критериями готовности приложения к выдаче заказчику выступают условия, отображенные в таблице 11.

Таблица 11 – Критерии приемочного тестирования

№	Критерий	Статус
1	В программе полностью отображено и доступно меню кафе и цены блюд	Готово
2	Функции добавления и удаления блюд исправно работают	Готово
3	Возможности подтвердить заказ и отправить чек на почту доступны	Готово
4	Обработчик ошибок своевременно выдает сообщения о некорректном использовании программы	Готово
5	Программа корректно высчитывает сумму заказа и показывает соответствующие акции	Готово
6	Возможность ознакомиться с информацией о программе доступна	Готово
7	Программа исправно закрывается при помощи специальной комбинации кнопок 	Готово
8	Успешное устранение важных / срочных дефектов программы	Готово

Заказчик готов будет принять приложение в случае, если отсутствуют дефекты приоритета критичный (Critical), значительный (Major), средний

(Normal), блокирующий (Blocker), однако допустима выдача приложения заказчику при наличии дефектов с приоритетом низкий (Low).

Таким образом, для эффективности реализации обеспечения качества программы «Menu 1.0» должны применяться различные методы тестирования, которые позволяют обнаружить дефекты, упущенные при разработке или не предусмотренные в логике обработке запросов и кода приложения, а также предоставить заказчику отчет о проведенном тестировании.

3.3 Оценка качества релиза приложения «Menu 1.0»

Полноценная оценка качества релиза приложения «Menu 1.0» реализуется после этапа разработки программного кода на основе подготовленной технической документации проекта и, в частности, плана и стратегии тестирования.

Рассмотрим модель «как есть» приложения «Menu 1.0» по методологии BPMN на рисунке 11. На основе данной модели можно сразу увидеть, что бизнес-процесс предприятия общественного питания нуждается в автоматизации деятельности, так как множество задач процесса ложится на обслуживающий персонал и требует много времени.

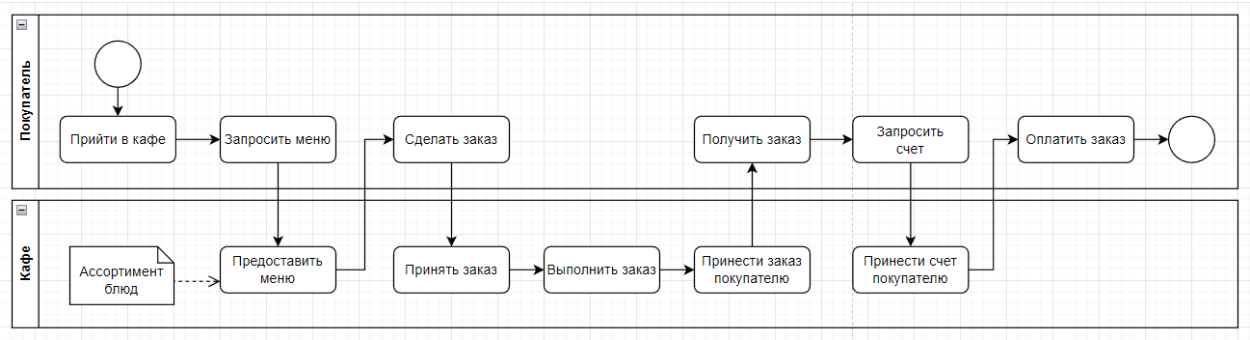


Рисунок 11 – Модель приложения «Menu 1.0»
по методологии BPMN «как есть»

Представим по методологии BPMN модель «как будет» приложения «Menu 1.0», изображенную на рисунке 12. Данная модель отражает процесс реализации бизнеса после внедрения приложения «Menu 1.0» в эксплуатацию.

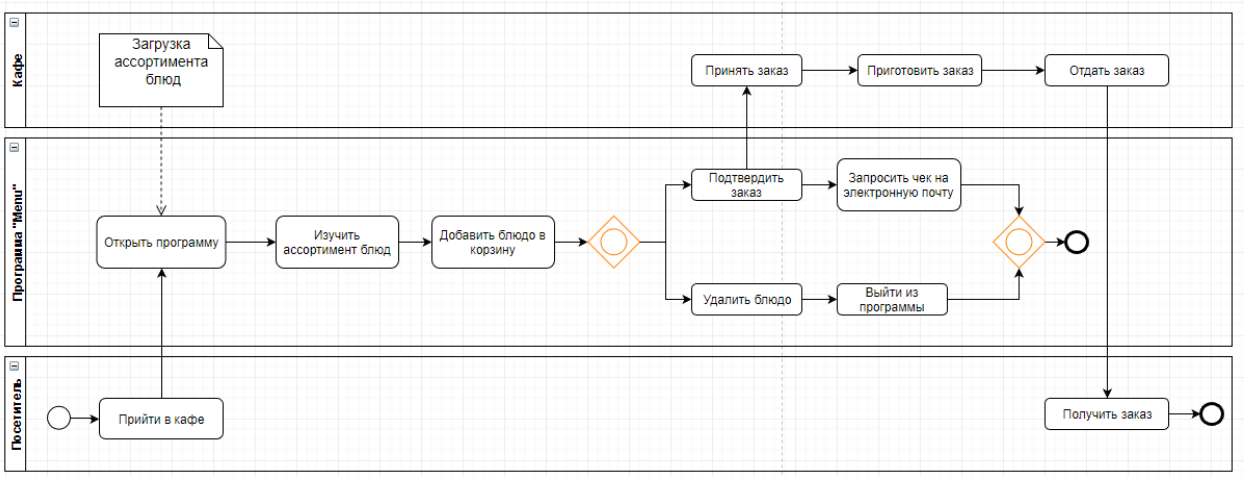


Рисунок 12 – Модель приложения «Menu 1.0»
по методологии BPMN «как будет»

Список и содержание тестовых сценариев для выполнения тестирования сформирован и отображен в плане тестирования. Каждому сценарию присвоен идентификационный номер, приоритет и описание цели проверки. Как и график проведения тестирования, тест-кейсы в технической документации международной IT-компании оформляются на английском языке (см. Приложение Б Таблица Б.1).

Итак, разберем разработанные функции приложения и сценарии работы пользователя с приложением «Menu 1.0».

Основное окно приложения «Menu 1.0» представлено как электронное меню кафе, где каждое блюдо доступно в определённое время. Товары, относящиеся к типу меню «Завтрак», доступны для выбора с 6:00 до 11:59:59, для товаров «Обеда» доступным временем является 12:00-17:59:59, а для выбора блюд из «Ужина» установлено время с 18:00 до 23:59:59.

Пользователь может выбрать одно из значений «Завтрак», «Обед», «Ужин» и перейти к выбору блюд. Заказ блюд осуществляется пользователем

при помощи листового списка с возможными вариантами для выбора, представленными на рисунке 13.

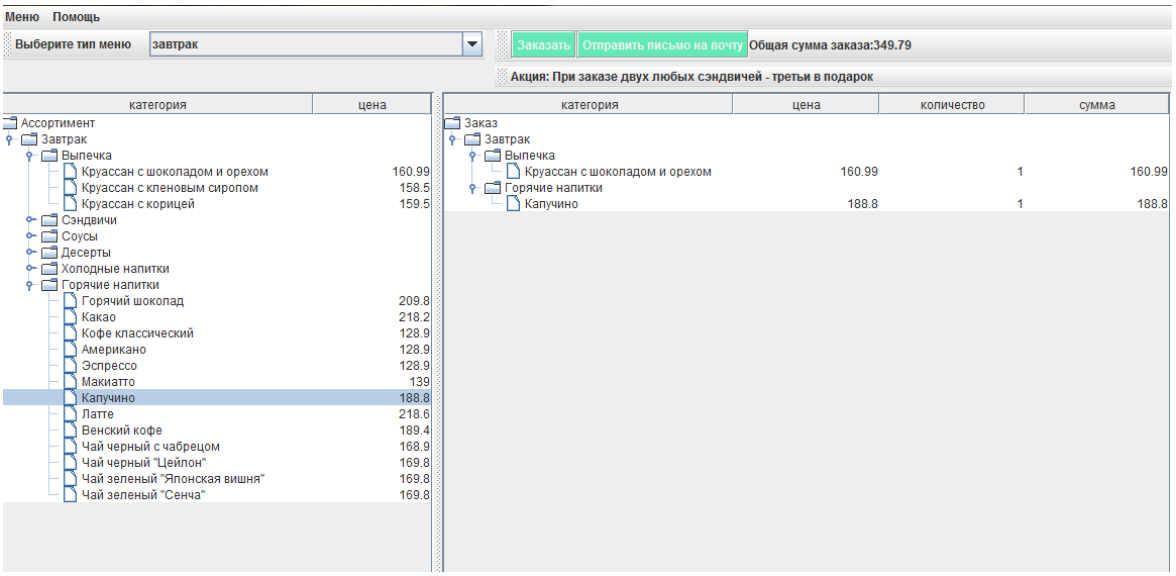


Рисунок 13 – Основное окно приложения «Menu 1.0»
с листовым списком товаров

В этом же окне, пользователь может посмотреть итоговую сумму своего заказа с учетом акций. Итоговая цена в чеке округляется до целого в пользу клиента, то есть в меньшую сторону.

Для удобства пользователя предусмотрена возможность добавления и удаления блюда из корзины. Добавление блюда в заказ осуществляется при помощи двойного клика на экране мобильного устройства, а удаление с помощью одинарного клика.

После окончания формирования заказа пользователь может подтвердить свой заказ. Данная возможность предоставлена через кнопку на главном окне [Подтвердить Заказ]. После нажатия данной кнопки открывается окно подтверждения, в котором пользователь может нажать [Подтвердить] или [Отмена]. После выбора кнопки подтверждения, появляется уведомление, показанное на рисунке 14, о принятом заказе.

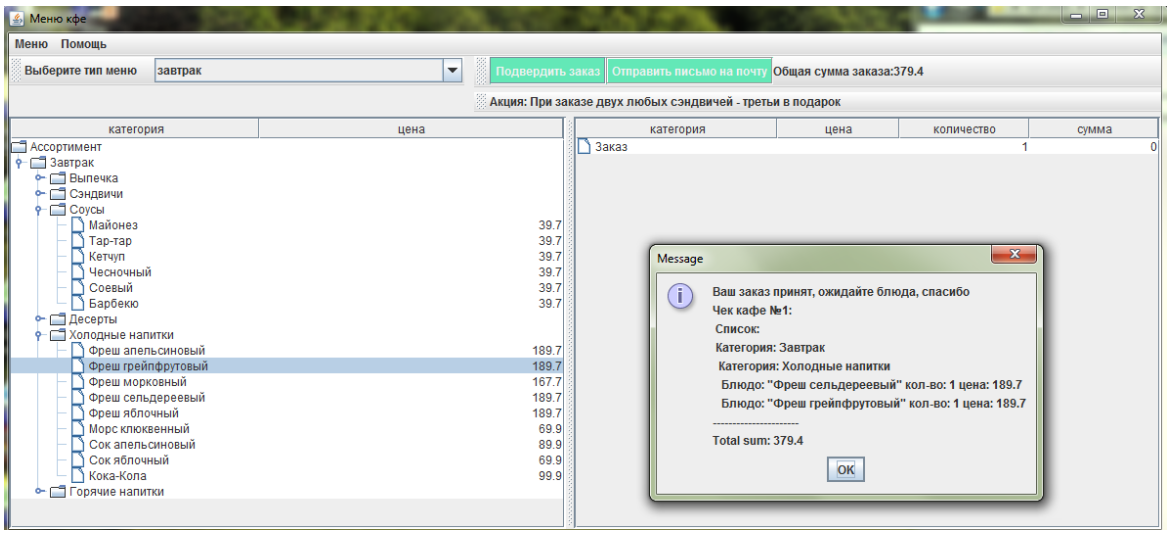


Рисунок 14 – Уведомление приложения «Menu 1.0» о принятии заказа

У пользователя приложения «Menu 1.0» также есть возможность отправить себе на электронную почту копию чека с информацией о выбранных товарах, учтенных скидках и сумме заказа. Для выполнения данной задачи в приложении после подтверждения заказа появляется дополнительное окно с кнопкой [Отправить чек на почту], изображенное на рисунке 15.

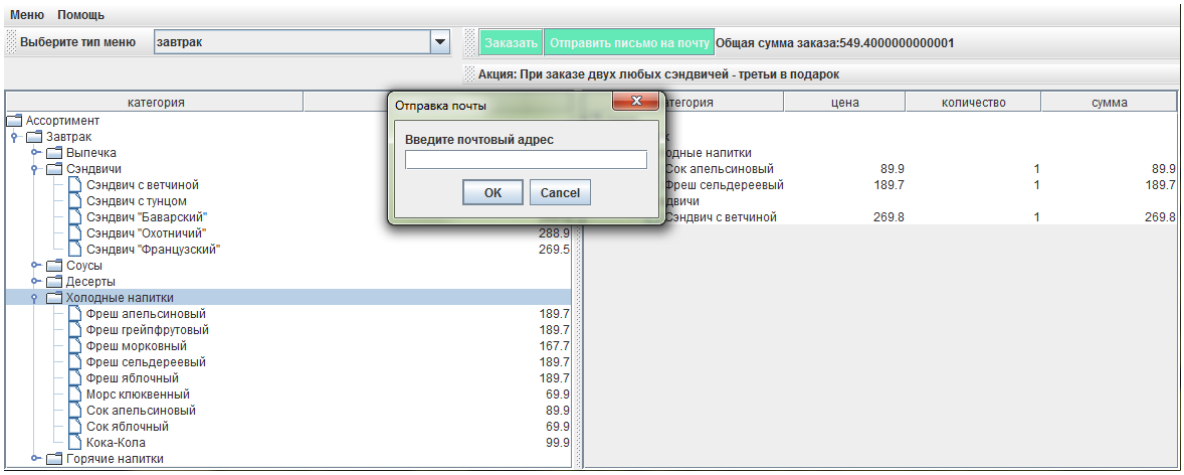


Рисунок 15 – Уведомление приложения «Menu 1.0»
об отправке письма на почту

Пользователь вводит в качестве адреса электронной почты любое значение, а приложение считывает введенный текст, проверяет на корректность и отправляет информацию. В случае ввода некорректного адреса приложение открывает дополнительное окно с конкретной информацией об ошибке, пользователь может ознакомиться и повторить попытку ввода.

Если пользовательский интерфейс не позволяет клиенту самостоятельно выяснить как пользоваться приложением, клиент может посмотреть инструкцию к приложению, которая содержит описание приложения «Menu 1.0», его назначение и версию путем нажатия кнопки на главном окне через бар-меню, кликом на пункт [Помощь]. Навигация к инструкции по пользованию приложения изображена на рисунке 16.

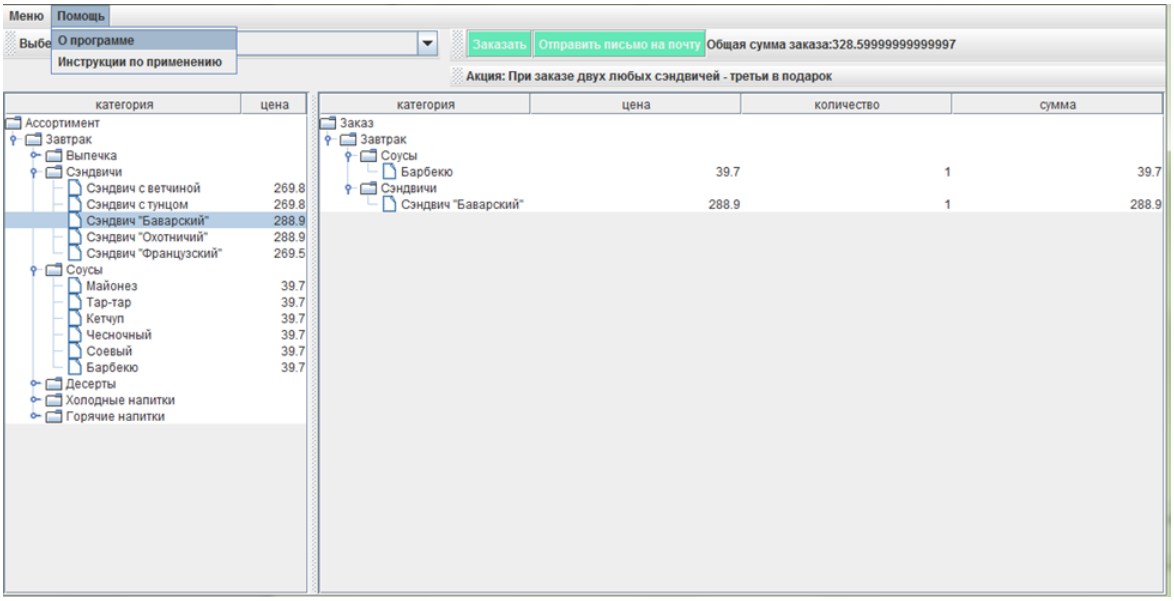


Рисунок 16 – Приложение «Menu 1.0» с навигацией на инструкцию

Для выхода из приложения пользователь может нажать кнопку «Выход» на главном окне через бар меню.

Для каждого типа меню в кафе действуют соответствующие акции, которые отображаются на главном окне. При соблюдении условий акции изменения применяются в итоговом чеке.

Рассмотрим пример таблицы 12, предоставленной на этапе разработки технической документации приложения «Menu 1.0». Данная таблица отражает возможность приложения «Menu 1.0» считывать размер скидки в зависимости от суммы, потраченной клиентом на заказ.

Таблица 12 – Расчет скидки в зависимости от потраченной суммы

Сумма в рублях	**Скидка в процентах**
0-500	1
500-3000	5
3000-100 000	15

В данном примере можно выявить дефект, связанный с неточностью эквивалентного класса. В случае, если пользователь осуществляет покупку на сумму ровно 500 рублей, то программа не сможет определить однозначный размер представляемой скидки 1 или 5 процентов. К тому же вероятно возникновение конфликтной ситуации с покупателем, если он получит только 1 процент скидки. Данный дефект возможно устранить путем установления более точных значений потраченной суммы, отраженных в таблице 13.

Таблица 13 – Расчет скидки в зависимости от потраченной суммы
с учетом эквивалентных классов

Сумма в рублях	**Скидка в процентах**
0-499,99	1
500-2999,99	5
3000-100 000	15

В результате исправления в технической документации грамотно отображаются три эквивалентных класса, которые позволят приложению «Menu 1.0» применять однозначную логику при вычислении размера скидки покупателю.

В процессе тестирования приложения необходимо проверять пограничные значения ввода данных. В приведенном выше примере

пограничными значениями будут выступать нижние и верхние пределы ввода суммы. Верхним пределом первого эквивалентного класса является сумма 499,99 рублей, а нижним пределом второго эквивалентного класса – 500 рублей.

Появление дефектов возможно как в спецификации, так и на этапе тестирования «сырого» приложения «Menu 1.0». Рассмотрим тестовый сценарий под идентификационным номером «TC01», указанный в плане тестирования и предназначенный для покрытия бизнес-процесса и проверки функции приложения «Добавление блюда в заказ клиента». Ожидаемым результатом являются два добавленных блюда в заказ с верным подсчетом чека. К нефункциональному тестированию можно отнести тест-кейс «Пользователю удобно отправлять готовую форму заказа» под идентификационным номером «TC23». Комфортный интерфейс для пользователя без видимых дефектов и сбоев работы приложения является ожидаемым результатом данного тест-кейса. В приемочном тестировании реализуются тест-кейсы на основе основных бизнес-требований заказчика. Например, сценарий «Чек клиента успешно отправляется в работу» с номером «TC25» основан на проверке работоспособности главной цели приложения, чем и является ожидаемый результат данного тестового кейса.

Итак, после прохождения тестовых сценариев в приложении «Menu 1.0» обнаружены определенные дефекты в разработанном решении, которые вносятся в отчет о дефектах приложения и передаются команде разработки на исправление кода. В итоговом отчете о найденных дефектах указываются идентификационный номер сценария, в котором обнаружен дефект, краткое описание дефекта, приоритет и серьезность дефекта, а также шаги воспроизведения сценария до момента получения ошибки, ожидаемый и фактический результаты. Данный отчет, отражающий результат реализации тестирования приложения «Menu 1.0», представлен в приложении А в виде таблицы А.1.

Результат прохождения этапа тестирования приложения «Menu 1.0» отражен на рисунке 17 в виде диаграммы в процентном соотношении.

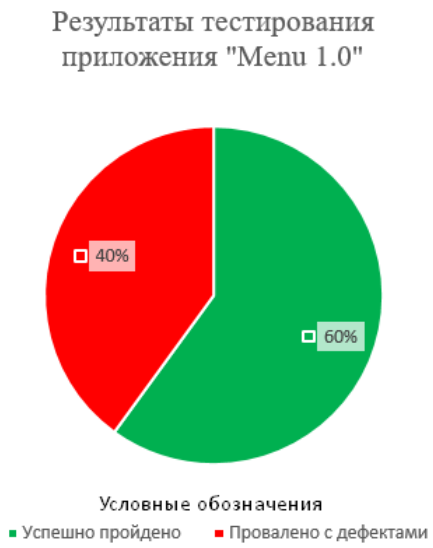


Рисунок 17 – Результаты тестирования приложения «Menu 1.0»

В связи с наличием дефектов, критичных для заказчика и потенциальных пользователей, в релизе 1.0 в приложении «Menu 1.0» сформирована диаграмма на рисунке 18, отражающая статистику актуальных дефектов в зависимости от приоритета.

Рисунок 18 – Статистика дефектов приложения "Menu 1.0" по приоритету

Для исправления кода и добавления кода дополнительной функции, например, поиска товара по вводимому тексту пользователем в поисковом поле, программист должен создать новую версию приложения «Menu 1.1», «Menu 1.2» и так далее. После успешного устранения дефектов, написания кода новой функции и ее интеграции с предыдущими функциональностями, специалистам по обеспечению качества необходимо сформировать и проверить дополнительные тестовые сценарии новой функции и провести регрессивное тестирование.

Регрессивное тестирование представляет собой повторную проверку новой версии приложения, предназначенную для подтверждения того, что внесенные изменения и добавления в код не повлияли на ту часть проекта, в которой не изменялся код. Регрессивное тестирование рекомендуется проводить каждый раз после корректировки программы или приложения, которая может включать исправление дефектов, слияние кода, миграцию на другую операционную систему или базу данных, добавление новой функциональности и другие изменения [21].

Таким образом, на основе сформированной технической документации разрабатано и проверено приложение «Menu 1.0». Для тестирования функциональной и нефункциональной областей приложения пройдены подготовленные тестовые сценарии и выявлены дефекты, не допускающие версию в релиз на эксплуатацию пользователям. Необходимая доработка и устранение дефектов переходит к команде разработки и затем выдается в релиз 1.1. Отчет по найденным дефектам передается заказчику для подтверждения результатов тестирования. После повторной выдачи релиза команда по обеспечению качества, в первую очередь, проверяет тестовые сценарии, в которых были найдены дефекты, затем остальные сценарии, не затронутые исправлениями. Регрессивное тестирование позволит своевременно выявить и устранить повторные или последующие за исправлениями баги до выдачи версии в релиз пользователям.

Заключение

Практические методы и инструменты оценки качества релиза программного обеспечения рассмотрены в магистерской работе на примере реализации приложения «Menu 1.0» через призму знаний, полученных при прохождении преддипломной практики в компании ООО «НетКрэкер». В ходе проведенного исследования были изучены концепции оценки и обеспечения качества программного продукта, разработанного при прохождении основных этапов каскадной методологии разработки, а также проанализированы распространенные методы тестирования и классификация дефектов, найденных по результатам реализации тестирования.

В ходе исследования были выявлены и сформулированы следующие выводы:

– тестирование как этап разработки в каскадной модели является ключевым, потому как именно команда по обеспечению качества может сделать вывод о соответствии программного обеспечения требованиям заказчика и выдвинуть решение о допуске программного обеспечения к эксплуатации. Процесс тестирования позволяет избежать негативных последствий как для проекта, так и для бизнеса, а также обеспечить соответствующее качество, исключая человеческий фактор в лице команды разработчиков бизнес-аналитиков;

– тестирование представляет собой процесс на основе исследовательского метода, позволяющий проводить испытание программного обеспечения с целью реализации проверки соответствия подготовленных требований проекта с фактическим состоянием разработанного программного обеспечения, а также устранения найденных несоответствий. Следует учитывать, что процесс обеспечения качества и процесс тестирования отличаются базовыми концепциями. Концепция тестирования заключается в улучшении качества программного обеспечения путем обнаружения дефектов фактически разработанного продукта, а

концепция обеспечения качества направлена на улучшение процесса разработки продукта с первого этапа жизненного цикла с целью предотвращения возможных дефектов на этапе тестирования;

– важную роль в разработке и обеспечении качества проекта играет техническая документация. Спецификация представляет собой документ, в котором собраны детальные описания каждого модуля и каждой функции программы. В дизайне описываются расположения модулей, кнопок, ссылок, картинок, а также их определенные сочетания цветов и размеров, которые должны быть созданы в конечном итоге проекта. Вышеуказанная техническая документация формируется по требованиям заказчика проекта. Разработка требований к программному обеспечению — это процесс, состоящий из мероприятий, необходимых для создания и утверждения документов, содержащих системные, функциональные, пользовательские требования, а также бизнес-требования заказчика проекта;

– одним из общепринятых определений дефекта программного обеспечения является расхождение между фактически готовым проектом и его спецификацией, только при условии, что спецификация существует и её содержимое сформулировано детально и правильно. Основным критерием при определении бага выступает наличие или отсутствие реализации проектом требований, которые ожидают получить пользователи;

– существуют различные виды дефектов как со стороны самого решения и имплементации, так и сторонние дефекты тестовой среды, интеграции компонентов, инструментов автоматизированного тестирования или даже в технической документации. Каждый найденный баг обладает жизненным циклом в процессе устранения и должен быть описан максимально детально для дальнейшего устранения разработчиками;

– регрессивное тестирование программного обеспечения представляет собой повторную проверку уже подготовленного релиза после каждого создания новой версии, предназначенную для подтверждения того, что внесенные изменения и добавления в код не повлияли на ту часть проекта,

в которой не изменялся код. Чтобы гарантировать высокое качество выпускаемой версии программы при сохранении оптимальных сроков и стоимости, необходим продуманный подход для определения стратегии и достаточного регрессивного покрытия;

– с целью наглядного представления эффективности метода поиска всех возможных покрытий решений в программном обеспечении в данном исследовании был предоставлен пример кода и необходимые вычисления на основе формулы цикломатической сложности Маккейба;

– процесс формирования требований к приложению «Menu 1.0», реализованного для предприятия общественного питания, и планирование его тестирования в данной работе основаны на теоретических знаниях и практических навыках, приобретенных во время прохождения производственной практики в компании ООО «НетКрэкер»;

– для эффективности реализации обеспечения качества приложения «Menu 1.0» применены различные методы тестирования, благодаря которым обнаружены дефекты, упущенные при разработке или не предусмотренные в логике обработке запросов и кода приложения. Список и содержание тестовых сценариев для выполнения тестирования сформирован и отображен в плане тестирования;

– после прохождения тестовых сценариев в приложении «Menu 1.0» обнаружены определенные дефекты в разработанном решении, которые вносятся в отчет о дефектах приложения. В итоговом отчете о найденных дефектах указываются идентификационный номер сценария, в котором обнаружен дефект, краткое описание дефекта, приоритет и серьезность дефекта, а также шаги воспроизведения сценария до момента получения ошибки, ожидаемый и фактический результаты.

Таким образом, реализованные на этапе тестирования мероприятия позволили своевременно выявить и устранить недопустимые дефекты релиза приложения «Menu 1.0», а также предоставить результаты заказчику в виде отчетности.

Список используемых источников

1. Баркалов С.А., Азарнова Т.В., Полухин П.В. Управление процессом тестирования веб-приложений методом фаззинга на основе динамических байесовских сетей // Вестник ЮУрГУ. Серия: Компьютерные технологии, управление, радиоэлектроника. 2017. №2. 51-64 с.
2. Блэк Р. Ключевые процессы тестирования. Планирование, подготовка, проведение, совершенствование. «Лори»: 2006. 544 с.
3. Вигерс К., Битти Дж. Разработка требований к программному обеспечению. 3-е издание, дополненное. СПб.: «БХВ-Петербург», 2019. 736 с.
4. ГОСТ Р 53622-2009. Информационные технологии. Информационновычислительные системы. Стадии и этапы жизненного цикла, виды и комплектность документов. Information technologies. Information-computing systems. Life cycle stages and steps, kinds and completeness of the documents. ОКС 03.120.10: утв. и введен в действие Приказом Федерального агентства по техническому регулированию и метрологии от 15 декабря 2009 г. N 964-ст. // Консорциум кодекс.
5. ГОСТ Р 56922-2016/ISO/IEC/IEEE 29119-3:2013. Национальный стандарт Российской Федерации. Системная и программная инженерия. Тестирование программного обеспечения. ОКС 35.080: утв. и введен в действие Приказом Федерального агентства по техническому регулированию и метрологии от 18 мая 2016 г. N 333-ст. // Консорциум кодекс.
6. Канер С., Фолк Дж., Нгуен Е.К. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. К.: «ДиаСофт», 2001. 544 с.
7. Качество программного обеспечения (Software Quality) // Бизнес-Анализ в России. [Электронный ресурс]. URL: https://analytics.infozone.pro/software-quality/ (дата обращения: 22.10.2022).
8. Кон М. Пользовательские истории. Гибкая разработка программного обеспечения. М.: «Диалектика», 2020. 258 с.

9. Корнипаев И. Требования для программного обеспечения: рекомендации по сбору и документированию. М.: «Книга по требованию», 2013. 118 с.

10. Котляров В.П. Основы современного тестирования программного обеспечения, разработанного на C#: Учебное пособие. СПб.: СПбГПУ, 2016. 334 с.

11. Криспин Л., Грегори Дж. Гибкое тестирование. М.: «Вильямс», 2010.

12. Кулаков К.А., Димитров В.М. Основы тестирования программного обеспечения. Петрозаводск: «ПетрГУ», 2018.

13. Майерс Г., Баджетт Т., Сандлер К. Искусство тестирования программ. Спб.: «ДиаСофт», 2014.

14. Майерс Г.Дж. Надежность программного обеспечения. (Software Reliability. Principles and Practices, 1976 [перевод с англ.]). М.: «Мир».

15. Макконнелл С. Совершенный код. Мастер-класс. М.: «Русская редакция», 2017. 896 с.

16. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. СПб.: «Питер», 2018. 352 с.

17. Мартюков А. С. О необходимости разработки гибкого процесса тестирования интернет-приложений // Новые информационные технологии в автоматизированных системах. М.: 2015. №14. 243 с.

18. Оценка эффективности автоматизации тестирования. [Электронный ресурс]. URL: https://www.a1qa.ru/blog/otsenka-effektivnostiavtomatizatsii-testirovaniya/ (дата обращения: 24.09.2022).

19. Панфилова А.П. Деловая коммуникация в профессиональной деятельности. СПб.: «Знание», 2004. 492 с.

20. Паттон Дж. Пользовательские истории. Искусство гибкой разработки программного обеспечения. СПб.: «Питер», 2019. 390 с.

21. Савин Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах. М.: «Дело», 2007.

22. Соловьев С.В. Технология разработки прикладного программного обеспечения: учебное пособие для студентов высших учебных заведений. М.: Академия естествознания, 2011. 407 с.

23. Ambler S.W., Sadalage P. Database Refactoring: Evolutionary Database Design. Boston: Prentice Hall PTR, 2006.

24. AQA – система автоматизированного тестирования бизнесприложений. [Электронный ресурс]. URL: https://www.galaktika.by/aqa-cistema-avtomatizirovannogo-testirovaniya-biznesprilozhenij.html (дата обращения: 24.09.2022).

25. Azarnova T.V., Polukhin P.V., Barkalov S.A. Expanding the Functionality of Fuzzing of Web Applications Based on Dynamic Bayesian Networks. Scientific and Technical Information. Institute of Electrical and Electronics Engineers (IEEE): 2018.

26. Burnstein I. Practical software testing: a process-oriented approach. New York: «Springer», 2013.

27. Functional Testing: A Complete Guide with Types and Example. [Электронный ресурс]. URL: https://www.softwaretestinghelp.com/guide-to-functional-testing/ (дата обращения: 10.12.2022).

28. How to Write Test Cases: Sample Template with Examples. [Электронный ресурс]. URL: https://www.guru99.com/test-case.html (дата обращения: 18.04.2022).

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

30. Netcracker Technology. An NEC Company. [Электронный ресурс]. URL: https://www.netcracker.com/ (дата обращения: 11.03.2023).

31. Pugh K. Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration. «Аддисон-Уэсли Профессионал», 2011. 368 с.

32. Software Testing Fundamentals. [Электронный ресурс]. URL: http://softwaretestingfundamentals.com/ (дата обращения: 16.02.2022).

Приложение А

Отчет о дефектах приложения «Menu 1.0»

Таблица А.1 – Отчёт о найденных дефектах приложения «Menu 1.0»

TC ID	Краткое описание дефекта	Severity	Priority	Шаги, приводящие к дефекту	Ожидаемый результат	Фактический результат	Скриншот
TC06	Отсутствует сообщение о правильной отправке письма	Major	Critical	1. Откройте главное окно. 2. Добавьте два разных блюда из обеденного меню двумя нажатиями левой кнопки мыши. 3. Нажмите кнопку “подтвердить заказ”, а затем кнопку “ок”. 4. Нажмите кнопку “отправить на электронную почту”. 5. Введите адрес электронной почты и нажмите кнопку “Ок”.	На дисплее появится окно о правильной отправке.	Нет никакого сообщения о правильной отправке.	
TC08	Неверный подсчет скидки	Critical	Critical	1. Откройте главное окно. 2. Добавьте десерт из меню ужина двумя нажатиями левой кнопки мыши. 3. Нажмите кнопку “подтвердить заказ”, а затем кнопку “ок”.	В соответствии со скидкой в квитанции предоставляется бесплатный чай.	При заказе появился другой чай (не такой, как чай по скидке).	
TC09	Программа не закрывается кнопкой "выход"	Major	Major	1. Откройте главное окно. 2. Нажмите кнопку “меню” и кнопку “выход”.	Окно программы должно быть закрыто после нажатия кнопки.	Программа не была закрыта после нажатия кнопки.	
TC12	Отсутствует сообщение о неправильном адресе электронной почты	Critical	Critical	1. Откройте главное окно. 2. Добавьте два разных блюда из обеденного меню двумя нажатиями левой кнопки мыши. 3. Нажмите кнопку “подтвердить заказ”, а затем кнопку “ок”. 4. Нажмите кнопку “отправить на электронную почту”. 5. Введите неправильный адрес электронной почты и нажмите кнопку “ок”.	На дисплее появится сообщение о неправильном адресе электронной почты.	Нет никакого сообщения о неправильном адресе электронной почты.	

Продолжение таблицы А.1

TC ID	Краткое описание дефекта	Severity	Priority	Шаги, приводящие к дефекту	Ожидаемый результат	Фактический результат	Скриншот
TC13	Отсутствует сообщение о пустой строке электронного письма	Major	Normal	1. Откройте главное окно. 2. Добавьте два разных блюда из обеденного меню двумя нажатиями левой кнопки мыши. 3. Нажмите кнопку “подтвердить заказ”, а затем кнопку “ок”. 4. Нажмите кнопку “отправить на электронную почту”. 5. Не пишите что-либо в строке и нажмите кнопку “ок”.	На дисплее появится сообщение о пустом электронном письме.	Нет никакого сообщения о пустом электронном письме.	
TC14	Отсутствует сообщение о повторной отправке на электронную почту	Minor	Low	1. Откройте главное окно. 2. Добавьте два разных блюда из обеденного меню двумя нажатиями левой кнопки мыши. 3. Нажмите кнопку “подтвердить заказ”, а затем кнопку “ок”. 4. Нажмите кнопку “отправить на электронную почту”. 5. Введите правильный адрес электронной почты и нажмите кнопку “ок”. 6. Снова нажмите кнопку “отправить на электронную почту”. 7. Введите другой правильный адрес электронной почты и нажмите кнопку “ок”.	На дисплее появится сообщение о повторной отправке по электронной почте.	Нет никакого сообщения о повторной отправке на электронную почту.	-

Продолжение таблицы А.1

TC ID	Краткое описание дефекта	Severity	Priority	Шаги, приводящие к дефекту	Ожидаемый результат	Фактический результат	Скриншот
TC16	Строка "количество блюд" в заказе не позволяет самостоятельно изменить количество	Minor	Major	1. Откройте главное окно. 2. Добавьте одно блюдо из меню ужина двумя нажатиями левой кнопки мыши. 3. Попробуйте изменить количество блюд двумя нажатиями левой кнопки мыши (например, 3). 4. Нажмите кнопку “подтвердить заказ”, а затем кнопку “ок”.	В строке "количество блюд" есть возможность изменить количество. (количество блюд должно быть 3)	Строка "количество блюд" возвращается обратно и не позволяет изменить количество.	
TC18	Подсчитывает сумму в случае пустого заказа	Major	Critical	1. Откройте главное окно. 2. Нажмите кнопку “подтвердить заказ”, а затем кнопку “ок”.	На дисплее появится сообщение о пустом заказе.	Программа подсчитывает сумму, пока заказ пуст.	
TC23	Неудобное расположение и цвет кнопки "подтвердить заказ"	Trivial	Normal	1. Другой пользователь открывает главное окно. 2. Пользователь добавляет блюдо и напиток из меню завтрака двумя нажатиями левой кнопки мыши. 3. Пользователь нажимает кнопку “подтвердить заказ”, а затем кнопку “ок”.	Новый пользователь может быстро найти кнопку "подтвердить заказ".	Новый пользователь долго искал кнопку, предпочтительно использовать кнопку большего размера и яркости.	-
TC24	Недостаточно удобный процесс удаления посуды	Trivial	Normal	1. Другой пользователь открывает главное окно. 2. Пользователь добавляет 3 блюда из обеденного меню двумя нажатиями левой кнопки мыши. 3. Пользователь удаляет одно блюдо одним нажатием правой кнопки мыши. 4. Пользователь нажимает кнопку “подтвердить заказ”, а затем кнопку “ок”.	Новый пользователь может быстро понять, как он может удалить блюдо из заказа.	Новый пользователь долго искал функцию "удалить", предпочтительно использовать кнопку корзины в строке меню.	

Приложение Б

Тестовые сценарии этапа тестирования

Таблица Б.1 – Тестовые сценарии этапа тестирования

TC ID	Название сценария	Приоритет	Основная идея сценария
Функциональное тестирование			
TC01	Добавление блюда клиентом	1	Цель сценария - добавить два блюда в заказ, сделанный пользователем.
TC02	Отправить квитанцию на адрес электронной почты	1	Цель сценария - является отправка квитанции на адрес электронной почты клиента.
TC03	Открыть информацию о программе	1	Цель сценария - открытие информации о программе пользователем.
TC04	Удалить блюдо из заказа	1	Цель сценария - удаление блюд из заказа пользователем.
Позитивное тестирование			
TC05	Добавить 3 похожих блюда, нажав на меню	1	Цель сценария - добавить 3 похожих блюда при оформлении заказа пользователем.
TC06	Отправить квитанцию на верный адрес электронной почты	1	Цель сценария - отправка квитанции на верный адрес электронной почты клиента.
TC07	Добавление блюд из всех видов блюд (завтрак, обед, ужин)	1	Цель сценария - добавить 3 разных блюда из всех видов блюд.
TC08	Выбрать десерт в соответствии с акцией на ужин	1	Цель сценария - выбрать любой десерт из меню ужина и получить бесплатный чай при получении.
TC09	Закрыть программу с помощью кнопки «выход»	1	Целью данного сценария является закрытие программы пользователем с помощью кнопки «выход».
TC10	Выбрать 3 блюда из одного вида меню и полностью удалить их	2	Цель сценария - выбрать 3 блюда из ужина и удалить все выбранные блюда из заказа.
TC11	Отправить квитанцию на адрес электронной почты перед подтверждением	2	Цель сценария - отправить квитанцию на адрес электронной почты перед подтверждением заказа.
Негативное тестирование			
TC12	Отправить квитанцию на неправильный адрес электронной почты	1	Цель сценария - отправка квитанции пользователем на неправильный адрес электронной почты.
TC13	Отправить квитанцию на пустой адрес электронной почты	1	Цель сценария - отправка квитанции на пустой адрес электронной почты (без заполнения строки).
TC14	Отправить квитанцию на несколько разных адресов электронной почты	1	Цель сценария - отправка пользователем одной квитанции на два разных адреса электронной почты.
TC15	Пользователь сам выбирает десерт и чай на ужин из меню	1	Цель сценария - самостоятельно выбрать отдельно десерт и чай в соответствии с условиями скидки на ужин.
TC16	Увеличить количество блюд в строке заказа	2	Цель сценария - попытаться увеличить количество блюд или напитков в строке заказа.
TC17	Добавить блюдо без скидки	2	Цель сценария - добавление пользователем двух блюд без скидки.
TC18	Добавить и подтвердить пустой заказ (без блюд)	2	Цель этого сценария - добавить и подтвердить пустой заказ (без блюд).

Продолжение таблицы Б.1

Нагрузочное тестирование			
ТС19	Заказать все блюда из одного вида меню	1	Цель сценария - заказать все блюда из меню завтрака одним пользователем.
ТС20	Добавить 10 похожих блюд и 10 похожих напитков в одном заказе	2	Цель сценария - создать 10 похожих блюд и 10 похожих напитков в одном заказе одним пользователем.
Стресс-тестирование			
ТС21	Нажимать все кнопки почти одновременно	2	Цель сценария - нажать все кнопки почти одновременно во время одного заказа.
ТС22	Создать 5 заказов один за другим	2	Цель сценария - создать и отправить подряд 5 заказов одним пользователем.
Тестирование удобства пользователя			
ТС23	Другие пользователи пытаются сделать заказ самостоятельно	1	Цель сценария – другие пользователи могут самостоятельно оформить заказ.
ТС24	Другие пользователи пытаются самостоятельно удалить блюда из заказа	1	Цель сценария - другие пользователи могут самостоятельно удалить блюда из заказа.
ТС25	Другие пользователи пытаются сами отправить квитанцию на адрес электронной почты	1	Цель этого сценария - другие пользователи могут самостоятельно отправить квитанцию на адрес электронной почты.