ЗАХАРОВ ВИКТОР

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. ОСНОВЫ

Виктор Захаров

Тестирование программного обеспечения. Основы

«Автор»
2023

Захаров В. В.

Тестирование программного обеспечения. Основы /
В. В. Захаров — «Автор», 2023

Погрузитесь в увлекательный мир тестирования программного обеспечения вместе с книгой, которая является настоящим концентратом чистейших знаний для новичков и профессионалов! Автор делится секретами мастерства, подробно рассказывая о более 15 видах тестирования и более 20 методах проектирования тестов (техниках тест-дизайна). И это только вершина айсберга знаний, изложенных в книге. Вы будете поражены глубиной информации и открытием знаний собранных в одном месте, о которых даже не догадывались. Книга насыщена ценнейшими советами, основанными на практическом опыте. Многочисленные примеры помогут быстрее освоить представленный в книге материал. Вооружившись знаниями из этой книги, вы будете уверенно разбираться в нюансах тестирования программного обеспечения и с лёгкостью применять знания на практике! Книга может по праву считаться настольной книгой специалиста по тестированию.

Содержание

Виктор Захаров
Тестирование программного обеспечения. Основы

Захаров Виктор

Тестирование программного обеспечения. Основы

Настольная книга специалиста по тестированию

Первое издание

Бердск, 2024

Аннотация

Погрузитесь в увлекательный мир тестирования программного обеспечения вместе с книгой, которая является настоящим концентратом чистейших знаний для новичков и профессионалов! Автор делится секретами мастерства, подробно рассказывая о более 15 видах тестирования и более 20 методах проектирования тестов. И это только вершина айсберга знаний, изложенных в книге. Вы будете поражены глубиной информации и открытием знаний, о которых даже не догадывались. Книга насыщена ценнейшими советами, основанными на практическом опыте. Многочисленные примеры помогут быстрее освоить представленный в книге материал. Вооружившись знаниями из этой книги, вы будете уверенно разбираться в нюансах тестирования программного обеспечения и с лёгкостью применять знания на практике! Книга может по праву считаться настольной книгой специалиста по тестированию.

Предисловие

На момент написания данной книги я работаю в одной из крупнейших компаний на должности руководителя управления, отвечающего за качество программного обеспечения. За плечами у меня 17 лет опыта в области информационных технологий, из них 13 лет я руковожу подразделениями тестирования.

В процессе работы регулярно возникала необходимость подбора новых сотрудников. Проводя собеседования, я видел низкую квалификацию кандидатов и большие провалы в знаниях и навыках тестирования. На должность специалистов приходят плохо подготовленные кадры!

Многие проходят курсы по тестированию, которые в наше время не создаёт только ленивый. По заверениям многих учебных центров их обучающие курсы ведут сертифицированные специалисты (сертификат ISTQB[1]). Однако огромному их количеству рано учить, им надо учиться. Это я понял, когда знакомился с программами, материалами и лекциями преподаваемых курсов. Я был разочарован качеством!

Тогда я принял решение создать учебную программу, где все имеющиеся у меня знания будут чётко структурированы, где не окажется противоречий, где есть большой практический блок, способный помочь в обучении собственных сотрудников. После длительной кропотливой работы был разработан курс для начинающих специалистов по тестированию, содержащий фундаментальные основы по тестированию.

Создавался он около года: писались материалы, проводилась сверка со стандартами, разрабатывалось программное обеспечение для практической работы и так далее. В подготовке мне помогала супруга. Через год курс был готов. Новые сотрудники, приходящие в подразделение, начали обучаться, повышая квалификацию. Результат не заставил себя долго ждать.

Курс представлял собой концентрат чистейших знаний по тестированию. В ходе разработки было подготовлено и переработано множество материалов, и в процессе обучения сотрудников он постоянно совершенствовался. Мне пришла в голову мысль, что все имеющиеся у меня на руках и в голове знания и опыт необходимо передать всем желающим. Настал момент написания книги.

Сейчас вы читаете ту самую книгу, которая поможет вам постичь фундаментальные основы тестирования программного обеспечения. Написана она простым и доступным языком. Материал будет понятен даже неискушённому читателю. Книга интересна и полезна как новичкам, так и опытным специалистам – каждый найдёт для себя необходимые знания и советы!

Располагайтесь удобнее. Мы с вами начинаем постигать таинственный мир тестирования программного обеспечения. Вас ждут новые открытия и знания. В добрый путь.

[1] ISTQB – международная квалификационная комиссия по тестированию программного обеспечения.

История тестирования

История тестирования компьютерных программ началась в 1950-х годах, когда впервые появились компьютеры. В то время программисты[2] самостоятельно проверяли свои программы на работоспособность, чтобы убедиться, что они работают правильно. В эти годы появилось одно из определений: тестирование – это процесс проверки программы с целью демонстрации её правильной работы.

Первые серьёзные программы создавались для научных исследований и для нужд министерств обороны. Требовалась чёткая и бесперебойная работа, а также отсутствие ошибок. В связи с этим процесс решили формализовать и стандартизировать. Проверка работоспособности программ проводилась формализовано с фиксированием всех изучаемых данных и полученных результатов.

В 1960-е годы люди стремились охватить программы полностью, то есть проверять все возможные передаваемые программе данные и все варианты выполнения программ. К примеру, программа может складывать большие числа, и вместо проверки сложения нескольких чисел, пробовали все возможные варианты без исключения: 1 + 1, 1 + 2, 1 + 3… 2 + 10, 2 + 11… 1259 + 15, 1259 + 16… и так далее.

Миллионы проверок! Это оказалось нереально, поскольку существует много данных, которые необходимо вводить или передавать в программу, много вариантов обработки передаваемых данных, и на их проверку уйдут годы. Также в документах, где описано, как создавать программу, присутствовало большое количество ошибок, которые трудно было найти. Поэтому метод полной проверки отклонили и признали неработоспособным. Появился один из принципов тестирования – исчерпывающее тестирование невозможно.

До 1970-х годов проверка программы означала демонстрацию её правильной работы. Но это занимало много времени и не давало полной информации о качестве программы. Подход оказался неэффективным. В 1970-х годах произошли изменения: вместо демонстрации правильной работы программы использовали поиск существующих в ней ошибок. Удачной проверкой считалась та, с помощью которой обнаруживали ранее неизвестную ошибку. В эти годы появилось очередное определение тестирования – это процесс проверки программы с целью нахождения ошибок.

В 1980-е годы начинается формирование концепции тестирования, дошедшей до наших дней. Проверка программ включила в себя такое понятие как «предупреждение ошибок».

Предупреждение ошибок – это информирование о возможной ошибке до того, как она станет серьёзной проблемой.

Это помогает исправить ошибку прежде, чем программа попадёт к пользователям и станет критической. Одним из эффективных способов предотвращения ошибок является проектирование тестов. То есть перед тем, как проверять программу, необходимо продумать, какие проверки мы будем делать, далее зафиксировать их списком и только после этого проверять работоспособность программы по составленному списку проверок.

В эти же годы стало понятно: необходимо сформировать принципы и подходы тестирования. Это позволит решать конкретные задачи и формализует процесс, чтобы он стал управляемым, дающим возможность контролировать качество программ на протяжении всех этапов их создания. В дальнейшем мы познакомимся с каждым этапом.

---

[2] Программист – специалист, занимающийся программированием (написанием кода программы), то есть созданием компьютерных программ.

Тестирование, существовавшее до начала 1980-х годов, проверяло только собранные и работающие программы. Это означает, что проверялась программа, которую можно было запустить и работать с ней. Однако с течением времени специалисты по тестированию начали принимать участие на всех этапах её создания, что позволяло выявлять проблемы заранее и уменьшать сроки и стоимость разработки. Тестирование стало больше чем просто поиск ошибок или демонстрация правильной работы программы.

В 1990-е годы понятие «тестирование» означало не только проверку программы, но и планирование процесса проверки, создание, поддержку и выполнение тестов, а также окружений, в которых работала и проверялась программа. Тогда тестирование стало важной составляющей для поддержания и улучшения качества программ. Это дало развитие инструментам, используемым для поддержки процессов тестирования: многофункциональные системы и инструменты для автоматизации тестирования; инструменты формирования отчётов; системы написания и хранения тестов и проведения тестирования; системы для проверки работы программ под высокой нагрузкой.

В настоящее время тестирование является важным инструментом для гарантии качества программ и уверенности в их правильной работе. Оно продолжает развиваться и не стоит на месте. Инновации в данной области позволяют постоянно улучшать качество и надёжность программ. Сейчас специалисты по тестированию играют важную роль в процессе разработки программ.

Кто он – специалист по тестированию

Мы пользуемся множеством различных программ каждый день, даже не задумываясь об этом. Например, покупая что-то в магазине, используются кассовые аппараты; снимая наличные деньги, мы используем банкоматы; посещая различные сайты, мы используем компьютеры, все это работает благодаря специальным программам.

Кто-то разрабатывает их, а кто-то проверяет работоспособность – тестирует. Специалистов, тестирующих программы, называют «специалистами по тестированию» или «тестировщиками». Есть профессиональный стандарт, где указано чёткое наименование профессии – «специалист по тестированию», поэтому в данной книге будем использовать термин оттуда.

Те, кто никогда не сталкивались с тестированием, иногда заблуждаются, думая, будто специалисты по тестированию – это люди, которые в процессе работы бессмысленно нажимают различные кнопки в проверяемой программе. Это не так. Тогда кто же он – этот специалист по тестированию?

Этот человек проверяет программы и в процессе проверки проводит их глубокий анализ и исследование. Он имеет аналитический склад ума, постоянно развивает навыки логически мыслить и анализировать большой объём информации, прежде чем решать поставленные перед ним задачи. Благодаря гибкости ума он моделирует различные ситуации, в которых программа может работать. И если она работает не так, как должна, а её ожидаемое поведение обязательно закреплено в специальной документации, специалист по тестированию должен сообщить программистам об ошибке. Специалисту по тестированию в этот момент понадобится важный для его профессии навык – умение чётко формулировать мысли и грамотно доносить информацию до других. Зачем нужен этот навык? Если программист не поймёт, что именно работает неправильно, он не сможет исправить ошибку или потратит на поиск непозволительно много времени.

Однако на этом работа специалиста по тестированию не заканчивается. Он следит за тем, чтобы программу или оборудование, в которое встроена программа, было удобно использовать. Например, если кто-то разместит монитор банкомата на уровне колен человека, пользоваться им будет затруднительно. Тестировщик должен сообщать о подобных недочётах тем, кто отвечает за проектирование удобства использования, чтобы исправить проблему до того, как такой банкомат перешёл в массовое производство. Аналогичная ситуация и с программами. Если с интерфейсом существуют значительные проблемы в удобстве её использования, специалист по тестированию должен сообщать об этом.

Он хорошо знает, как работает программа, которую проверяет. Порой даже лучше программистов, которые её создают. Если программа очень большая, команда программистов делится на группы. Каждая знает только ту часть, за разработку которой отвечает, а тестировщикам приходится изучать и знать функционирование всей программы целиком, чтобы проверять её работу комплексно. По опыту могу сказать: программисты ценят квалифицированных специалистов по тестированию, и когда у новых программистов возникают вопросы связанные с работой программы, они идут к ним за помощью. Встречались ситуации, когда новые программисты или аналитики[3], изучая работу новой для них программы, обучались у опытных тестировщиков.

[3] Аналитик – это специалист, который занимается изучением информации и делает выводы на основании этой информации. Он анализирует данные, чтобы понять, как должна работать программа в конкретных ситуациях. Не полный перечень занятий аналитика: общение с пользователями; выявление требований пользователей к программе, которую будут разрабатывать; документирование требований и их согласование; постановка задач программистам; демонстрация готовой программы пользователям.

Надо помнить, что на плечи специалистов по тестированию ложится ответственность за качество работы программ. И от качества их работы зависит очень многое: от размера прибыли компании до жизни людей. Практически каждая компания, связанная с разработкой программ, имеет в своём штате тестировщиков.

Зачем нужны специалисты по тестированию

Никто не идеален, все мы допускаем ошибки. Они могут быть как незначительными, так и очень серьёзными, некоторые имеют разрушительные последствия. Поэтому, когда мы создаём какой-либо продукт, в том числе компьютерные программы, необходима проверка, чтобы его использование было безопасным и эффективным. В этот момент и нужны специалисты по тестированию.

У многих возникает вполне закономерный вопрос: «Зачем привлекать специалистов по тестированию для проверки программ, если это могут делать программисты, разрабатывавшие её?». В некоторых компаниях, где нет специалистов по тестированию, так и происходит. Программисты сами и разрабатывают, и тестируют. Однако не всё так просто.

Во-первых, если программист сам занимается тестированием своей программы, у него будет меньше времени, чтобы фокусироваться на прямых обязанностях: разработке программы и устранении ошибок. Таким образом, время, которое уходит на разработку и тестирование в целом, значительно больше, чем если тестирование выполняется специалистами по тестированию. Конечно же программист проверяет программу перед тем, как передать её специалистам по тестированию, однако делает это поверхностно, чтобы удостовериться, что логика, которую он реализовал, работает. Если он этого не сделает, и программа, переданная на тестирование, не будет функционировать, специалисты по тестированию вернут её на доработку.

Во-вторых, программист не всегда может предусмотреть все возможные способы использования программы, поскольку его мышление отличается от мышления специалиста по тестированию. Последний задумывается обо всех возможных способах использования программы, в том числе, учитывая и возможные варианты её сломать, так как пользователь программы, если у него что-то не получается, может начать щелкать все по очереди, надеясь, что что-то заработает. Программист обычно думает, как правильно использовать программу, и может не предусматривать случаи, когда программа может сломаться. В результате, такой подход способен привести к пропуску критических ошибок, которые могут обнаружить конечные пользователи.

В-третьих, специалист по тестированию, используя различные техники, методы и виды тестирования проводит более тщательную проверку, ведь это его профессия, он совершенствует навыки годами.

Отсюда и вывод: специалисты по тестированию нужны везде, где есть программисты. Программисты разрабатывают, а специалисты по тестированию – тестируют.

Что мы знаем о программах

В предыдущих главах мы упоминали программы, которые разрабатываются и тестируются. В жизни вы слышали термин «программное обеспечение». Кто работает в сфере информационных технологий, могли ещё слышать термин «информационная система». Многие считают, что программа, программное обеспечение и информационная система – это одно и то же. Давайте попробуем открыть завесу тайны и понять, что означают данные термины.

Начнём с «программы». Сначала приведём общепринятое понятие.

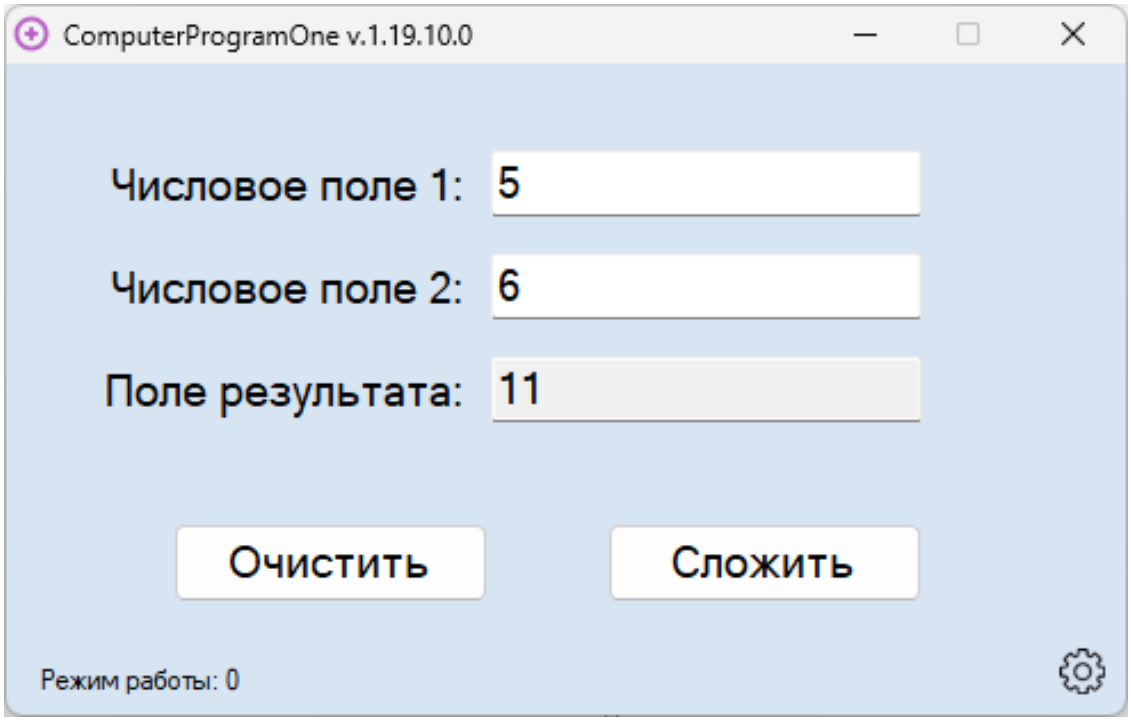
Программа – это набор инструкций, написанных на определённом языке программирования, которые компьютер может исполнять, для выполнения определённой задачи.

Простыми словами: программа представляет собой рецепт для компьютера. Когда мы готовим еду, нужен рецепт, чтобы знать, что и как делать. То же самое и с компьютером – ему необходима программа, чтобы выполнить определённую задачу. Рецепт содержит инструкции, как приготовить блюдо. Аналогично и программа содержит программные инструкции в коде. Но в отличие от рецепта, компьютерная программа исполняет инструкции автоматически или при определённом условии. Это значит, что, когда мы запускаем программу на компьютере, она самостоятельно выполняет определённые инструкции, прописанные в ней, или выполняет прописанные инструкции при определённом действии пользователя.

Рассмотрим пример простой программы, где есть две инструкции:

1) При нажатии на кнопку «Сложить» выполнить операцию сложения чисел и вывести результат сложения.

2) При нажатии на кнопку «Очистить» очистить все имеющиеся у программы поля.



Данная программа для сложения чисел будет сопровождать нас на протяжении всей книги, поэтому можете скачать её для ознакомления на сайте автора[4].

Переходим к термину «программное обеспечение».

Программное обеспечение (ПО) – совокупность программ, используемых для управления компьютером.

Исходя из определения можно сказать, что программное обеспечение – это набор программ, установленных на компьютере и предоставляющих возможности для выполнения различных задач. Данный набор включает все программы на компьютере или устройстве, необходимые для их работы.

Теперь мы понимаем, что программа и программное обеспечение – это два разных понятия. Программа – часть программного обеспечения, а программное обеспечение – совокупностью всех программ, установленных на компьютере.

Вы часто будете сталкиваться с ситуациями, когда под словами «программное обеспечение» люди подразумевают программу. Так повелось, и на это надо реагировать спокойно.

Теперь нам предстоит понять, что такое «информационная система».

Информационная система – это комплекс программ и устройств, которые работают вместе для сбора, обработки, хранения, предоставления и передачи информации с целью решения определённых задач.

Информационные системы могут состоять из программ, компьютеров, сети передачи информации, баз данных[5], устройств ввода-вывода[6] и так далее. Цель информационных систем – облегчение, повышение эффективности и производительности процессов. Они используются для решения широкого спектра задач.

Теперь рассмотрим все три понятия в связке на простом примере. Мы покупаем компьютер. На нём ничего не установлено, и это является оборудованием. Далее устанавливаем операционную систему, которая включает набор различных программ: калькулятор, редактор текста и т. д. Теперь у нас есть компьютер с программным обеспечением.

Мы разрабатываем программу, в которой можно заполнять налоговые отчёты. Она установлена на первом компьютере. На втором устанавливаем базу данных, в которой наша программа хранит все данные включая созданные отчёты. На третий устанавливаем программу, которая получает из базы данных информацию по отчётам и автоматически отправляет их в налоговые органы. Всё описанное является информационной системой, включающей три компьютера, без которых не будут функционировать программы; программы для создания отчётов, их отправки, база данных; программное обеспечение с операционными системами со всеми перечисленными программами; сеть, которой связаны компьютеры и по которой они обмениваются информацией.

Есть ли элемент меньше программы? Есть – программный компонент.

Программный компонент – это автономный наименьший элемент программы, который создан для выполнения конкретных функций или задач.

---

[4] Программа «Computer Program One» для Windows размещена для скачивания по ссылке https://victorz.ru/books/book-1

[5] База данных (БД) – это хранимый набор данных, который каким-либо образом структурирован.

[6] Устройства ввода-вывода – это устройства, которые позволяют нам говорить, вводить информацию в компьютер и получать информацию от компьютера. Клавиатура и мышь – это устройства ввода, потому что мы с их помощью вводим информацию в компьютер. Экран и принтер – устройства вывода, потому что выводят информацию из компьютера.

Можно сказать, что программные компоненты – кирпичики, из которых строятся программы. Как и в случае с настоящими кирпичиками, программисты могут использовать различные готовые компоненты в своих программах, чтобы не приходилось писать новый программный код с нуля.

Рассмотрим на примере. Представим, что нам необходимо создать программу для рисования. Вместо того, чтобы писать код для каждой функции программы (например, для создания линий, окружностей, прямоугольников), можно использовать готовые компоненты, созданные не нами, которые уже выполняют эти функции. Это бывает как стандартный набор графических компонентов, предоставляемый операционной системой, так и специализированные компоненты, которые мы можем загрузить из интернета.

Другой пример: создание онлайн-магазина. Вместо того, чтобы писать код для каждой функции, такой как добавление товаров в корзину, оформление заказа, обработка платежей и отправка уведомлений покупателям, мы можем использовать готовые компоненты, в которых уже реализованы данные функции.

В процессе чтения книги вы будете сталкиваться со всеми рассмотренными понятиями: программа, программное обеспечение, информационная система, компонент. Это сделано для того, чтобы вы привыкали к данным понятиям. Они будут использоваться, только если это уместно в определённом контексте.

Клиент-серверная архитектура программ

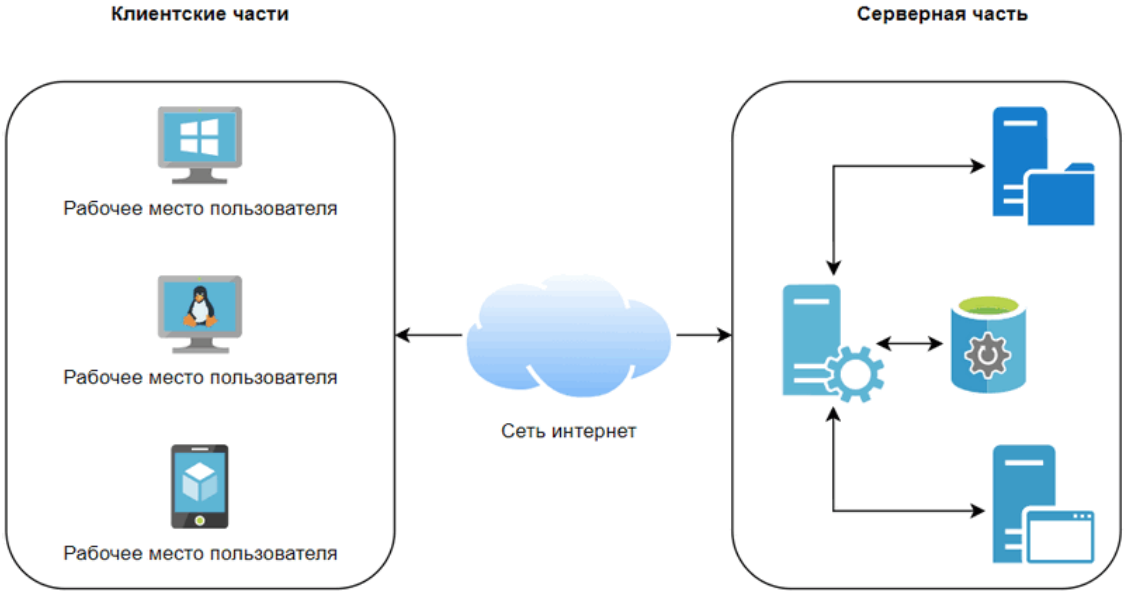
В разделе, описывающем виды тестирования, мы столкнёмся с понятиями клиент-серверной архитектуры. В связи с этим разберём, что это такое. Также данная информация будет вам полезна для общего понимания работы ряда информационных систем.

Клиент-серверная архитектура – это принцип построения информационных систем и программ, которые работают через интернет или локальную сеть.

Это как разделение обязанностей между компьютерами. В данной системе есть два типа компьютеров: «клиент» – клиентская часть, «сервер» – серверная часть. Они являются двумя составными частями информационных систем и программ, работающих по сети. Давайте посмотрим на них на примере интернет-магазина.

Клиентская часть – то, что работает у пользователя на компьютере или телефоне. В нашем случае – сайт или программа интернет-магазина, его интерфейс, где отображается каталог товаров, корзина, личный кабинет.

Серверная часть – это программа или набор программ, установленных на удалённом компьютере или компьютерах организации. Там хранятся все данные о товарах, заказах, пользователях. Туда приходят ваши запросы из клиентской части при нажатии на кнопки, а обратно с сервера приходят данные в клиентскую часть для отображения.



Клиентская часть информационной системы – это то, что использует пользователь напрямую, а серверная работает на стороне организации, обеспечивая все основные функции. Таким образом, первая ориентирована на взаимодействие с пользователем, в то время как вторая заботится о более сложных задачах и обеспечивает функциональность, которую видит пользователь в клиентской части. Вместе они образуют работающую информационную систему.

Программа для сложения чисел, текстовый редактор не имеют клиент-серверной архитектуры и их можно назвать «автономными программами». Им не требуется для работы серверная часть. Интернет-магазины, мобильные программы различных магазинов, сетевые игры

– информационные системы с клиент-серверной архитектурой. Для их работы нужна серверная часть.

Жизненный цикл программы

Как любая сущность в мире, программы появляются на свет, «живут» определённое время и «умирают», и на протяжении этого периода проходят различные стадии своего существования.

Идея. Появление программы начинается с неё. Есть заинтересованное лицо или группа лиц, которые в определённый период времени понимают: нужна программа, которая будет помогать им решать определённые задачи. В момент появления идеи и принятия решения о её создании начинается «жизнь» любой программы.

Анализ. На данном этапе определяются нужды и ожидания пользователей будущей программы и собираются их пожелания. По мере того, как выясняются потребности, они преобразуются в требования, которым должна удовлетворять новая программа. Они чётко, структурировано и формализовано фиксируются в документе. О требованиях поговорим в следующих главах.

Проектирование. На этапе анализа определяется, что должна делать будущая программа; определяется, как она будет выполнять свои задачи и какие функции необходимо заложить. Именно на данном этапе определяется структура программы, базы данных, как программа будет взаимодействовать с другими программами, как будет выглядеть интерфейс и прочие моменты.

Разработка. На этом этапе происходит написание кода, создание файлов данных и разработка баз данных.

Тестирование. В этот период проводятся исследования и испытания программы. Данный этап тесно связан с разработкой, так как в процессе разработки идёт постоянное тестирование.

Внедрение. Происходит ввод программы в эксплуатацию и передача её пользователям.

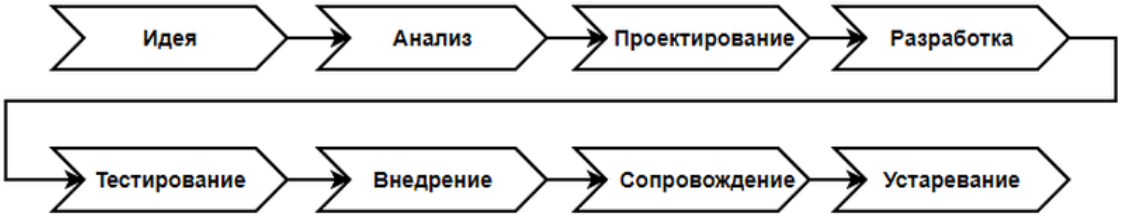
Сопровождение. Программа поддерживается и обновляется, чтобы оставаться актуальной и исправно работающей.

Устаревание. На этом этапе программа выводится из эксплуатации из-за устаревания или замены на более новую версию. Прекращается сопровождение и разработка, завершается «жизнь» программы.

Рассмотрим все описанные этапы на примере строительства. Изначально у человека появляется идея постройки дома. Через определённое время он начинает проводить анализ и задаваться вопросами: «сколько комнат должно быть в доме», «нужна ванна или душевая кабинка», «какой фундамент выбрать», «сколько должно быть этажей», а также он спрашивает о потребностях свою семью. Это и есть период анализа, сбора требований и пожеланий. Организованный человек все пожелания фиксирует на бумаге, и у него появятся зафиксированные требования к дому. После анализа человек идёт к архитекторам, чтобы они разработали ему индивидуальный проект под его требования. Архитекторы начинают проектирование дома, коммуникаций, придомовой территории. Дальше человек получает на руки всю проектную документацию и направляется к строителям, начинается разработка дома – строительство. Затем он принимает дом и проводит тестирование: крутит краны, топает по полу, изучая прочность, проверяет систему отопления, кондиционирования. Если всё удовлетворяет, он въезжает, и этап переезда – это внедрение. Далее уже идёт эксплуатация – период проживания. Если строители дали на дом гарантию или на платной основе готовы постоянно поддерживать его в хорошем состоянии, в случае поломки чего-либо, человек обращается в организацию, строившую дом, и они исправляют проблему, а возможно что-то совершенствуют – это сопровождение. Через много лет дом станет непригодным для проживания – тогда его выведут из эксплуатации и снесут; на этом этапе завершается жизнь дома – этап устаревания.

Теперь возьмём пример операционной системы. В далёкие времена у кого-то появилась **идея** создания операционной системы Windows. Вдохновлённый ею человек начал проводить **анализ** существующих программ на рынке и собирать техническую информацию. Накопив необходимые данные стало понятно – нужно создавать. После началось **проектирование** операционной системы – продумывалось, из каких программ она будет состоять, как программы будут между собой взаимодействовать, как будет выглядеть интерфейс операционной системы. Пройдя этап проектирования, началась **разработка** – написание кода операционной системы. Создав первую версию, программисты с коллегами начали **тестирование** работы операционной системы и её компонентов. Убедившись, что операционная система работает, провели **внедрение** – передали пользователям и научили их работать с операционной системой. Люди пользовались операционной системой, обнаруживали ошибки и сообщали создателям. Те, в свою очередь их исправляли и обновляли операционную систему. Это этап **сопровождения**. Создатели выпустили новую версию, а затем ещё одну. Первую версию вывели из эксплуатации, так как произошло **устаревание,** её больше не поддерживали. Так и закончилась жизнь первой версии операционной системы. Это применимо ко всем существующим программам.

Описанные стадии жизни программы и есть её жизненный цикл, состоящий из этапов:



При этом, пока не произошло вывода из эксплуатации (устаревания), процесс цикличен, т. е. все этапы кроме «Устаревания» многократно повторяются.

Жизненный цикл программы – период времени, который начинается с момента принятия решения о необходимости создания программы и заканчивается в момент её полного изъятия из эксплуатации.

Требования к программе

Мы с вами получили представление о жизненном цикле программ. В процессе его рассмотрения мы затронули такое понятие как «требования». Рассмотрим, что они из себя представляют, для чего нужны и откуда берутся.

Чтобы разработать программу, необходимо понять, что пользователь хочет получить от неё, какие нужды хочет закрыть. На основании полученной информации аналитики начинают подробнее разбирать потребности пользователей и фиксировать, как должна работать программа, что должна делать, с какими программами должна взаимодействовать, а также прописывают многие другие аспекты на основании постоянного общения с пользователями и выявления их потребностей и желаний. Все данные фиксируются в виде требований к программе. Из озвученного выведем определение, что такое «требование к программе».

Требование к программе – это структурированное описание определённых свойств программы (поведения, внешнего вида, качества и т. д.), которые должны отвечать потребностям пользователя. Требования могут представляться в виде документа или набора документов.

К примеру, в ходе общения с пользователями аналитик выясняет, что кнопка в программе должна быть зелёная – это требование. Программа должна запускаться за 5 секунд – это требование. Итого мы уже имеем два требования.

Они фиксируются в документах, которые имеют своё название – «спецификация».

Спецификация – документ, исчерпывающе, однозначно и доступно описывающий требования, дизайн, поведение и иные характеристики программы, которую требуется разработать.

Спецификацию ещё могут называть «спецификация требований». Ниже приведена часть спецификации с таблицей требований к программе:

Номер	Название	Требование
ФТ-1.1	Появление формы	Форма появляется при запуске программы. Форма появляется по центру экрана.
ФТ-1.2	Масштабирование формы	Размер формы не изменяется (фиксированный).
ФТ-1.3	Перемещение формы	Форма перемещается по экрану, если навести курсор мыши на заголовок формы, зажать левую кнопку мыши и произвести перемещение формы.
ФТ-1.4	Поле ввода первого числа	В поле можно вводить только положительные целые числа. Максимальное количество вводимых и отображаемых символов 4 (четыре). В поле можно вводить информацию с клавиатуры. В поле можно вставлять данные из буфера обмена. Из поля можно копировать данные в буфер обмена.
ФТ-1.5	Поле ввода второго числа	В поле можно вводить только положительные целые числа. Максимальное количество вводимых и отображаемых символов 4 (четыре). В поле можно вводить информацию с клавиатуры. В поле можно вставлять данные из буфера обмена. Из поля можно копировать данные в буфер обмена.

Спецификации, которые составляют аналитики, от компании к компании отличаются, нет чёткого шаблона, которого придерживаются все аналитики мира.

Она может содержать функциональные и нефункциональные требования – это два основных вида требований к программному обеспечению. В чём их отличия?

Функциональные требования – описывают функциональность, предоставляемую программным обеспечением. Они определяют, что должна делать программа. Сюда входят: возможность ввода и редактирования данных в поле программы; возможность очистки полей для ввода данных; возможность сложения чисел и т. д. На рисунке выше как раз отражена спецификация с функциональными требованиями.

Нефункциональные требования – описывают свойства программы, но не её поведение. Они определяют, как должна работать программа. Сюда относятся: удобство использования (к примеру, расположение кнопки или её название); производительность; безопасность и т. д. Ниже приведена спецификация с нефункциональными требованиями:

Номер	Название	Требование
НТ-3.1	Название файла	Файл логирования имеет имя «ComputerProgramOne.log».
НТ-3.2	Путь к файлу	Файл логирования располагается рядом с исполняемым файлом программы.
НТ-3.3	Формат файла	Файл имеет текстовый формат и хранит данные в соответствующем формате

Основное отличие заключается в том, что функциональные описывают функционал, нефункциональные – свойства и качества системы.

В ходе своей профессиональной деятельности специалисты по тестированию часто сталкиваются с ситуацией, когда в организации отсутствуют чётко задокументированные требования к программе или они являются минимальными и не полностью информативными. В таких случаях специалисты по тестированию вынуждены активно взаимодействовать с аналитиками и программистами, чтобы получить необходимую информацию. Иногда им даже приходится самостоятельно проводить исследование программ, чтобы полноценно провести тестирование. Это требует от специалиста по тестированию гибкости и способности самостоятельно разбираться в функциональности программ.

Дефект, ошибка и отказ

Собрав информацию о потребностях пользователей и проработав требования, все данные передаются программистам. Они, изучив их, начинают разрабатывать программу, реализовывая то, что зафиксировано в требованиях. В процессе написания кода программисты могут допустить ошибку и прописать неверные данные в коде. К примеру, разрабатывая программу сложения чисел программист в строке, где должно происходить сложение, вместо знака «плюс» прописал знак «минус». Так была допущена ошибка:

```
private void МетодСложенияЧисел()
{
    int _первоеЧисло = Int32.Parse(полеВводаПервогоЧисла.Text);
    int _второеЧисло = Int32.Parse(полеВводаВторогоЧисла.Text);
    int _суммаЧисел = _первоеЧисло - _второеЧисло;
    полеОтображенияРезультата.Text = _суммаЧисел.ToString();
}
```

Ошибка – действие человека, которое приводит к неправильному результату.

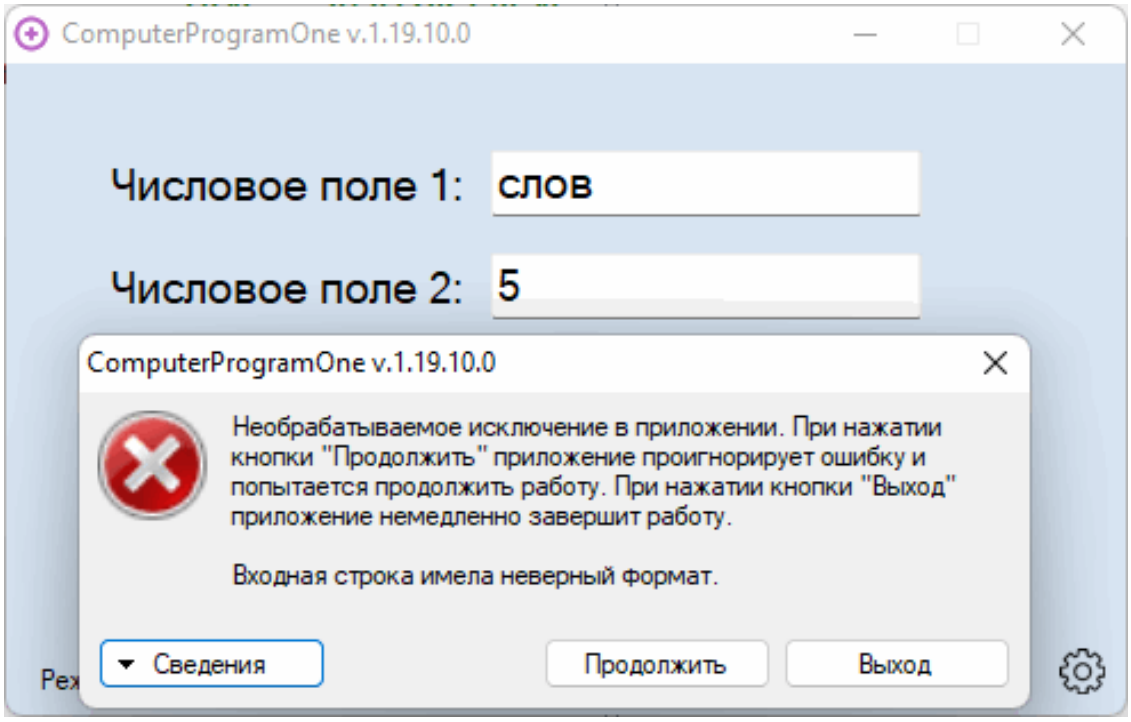
После написания кода программы её специальными инструментами «собирают», чтобы она стала полноценной работающей программой, а не набором файлов со строками кода. Собрали, внедрили и начали процесс эксплуатации. В момент использования программы пользователи столкнутся с ситуацией, когда программа работает не так, как ожидалось, когда реальный результат не равен ожидаемому.

Если рассматривать приведённый пример с программой сложения чисел, пользователь ожидает, что, введя два числа и нажав на кнопку сложения, получит результат сложения. Однако его ждёт разочарование, так как программа будет вычитать и выводить результат вычитания, а это означает, что она не сможет выполнять требуемую от неё функцию. Почему? Потому что на этапе разработки допустили ошибку, и пользователь столкнулся с последствиями ранее допущенной ошибки – с дефектом в программе.

Дефект – изъян в программе, который может привести к невозможности выполнить требуемую функцию.

Чтобы дефекты не попадали к пользователям, они должны выявляться специалистами по тестированию до передачи программы в эксплуатацию.

В ходе работы программы дефекты могут привести к отказу. Пример: программист, создавая программу, не добавил в код проверку, запрещающую программе складывать текст с числами. В этом случае появится дефект, который приведёт к отказу программы, и она попытается аварийно завершить свою работу:



Отказ – нарушение работоспособности программы, при котором она перестаёт выполнять целиком или частично свои функции.

В итоге получаем следующую последовательность: допущенная программистом в коде ошибка приводит к появлению в программе дефекта, и при работе он может повлечь отказ программы. В свою очередь отказ может привести к большим финансовым потерям компании или к катастрофе, если речь идёт об атомных реакторах или самолётах. Всё зависит от информационной системы, в которой произошёл отказ.

Дефекты могут появляться не только из-за допущенных в коде программ ошибок, но и из-за ошибок в настройках. К примеру, у нас есть требования, что в интернет-магазине заказы могу оплачивать все категории покупателей. Настраивая интернет-магазин, специалист по невнимательности указал, что оплачивать заказы могут только пользователи, которые не вошли в систему (анонимные пользователи). В этом случае не смогут оплачивать заказы те, кто вошли в систему (авторизованные пользователи). В итоге допущенная в настройках ошибка привела к появлению дефекта (невозможно выполнить требуемую функцию) и отказу системы (программа перестаёт выполнять свои функции) при оплате заказов авторизованными пользователями.

Критичность и приоритет

Мы определили следующее: чтобы дефекты не попадали к пользователям, они должны выявляться специалистами по тестированию до передачи программы в эксплуатацию. Если на ранних этапах дефекты не обнаружатся и попадут с программой к пользователям, то они будут оказывать влияние на работу самих пользователей. К примеру, бухгалтер на неработающем программном калькуляторе не может складывать числа, а это влияет на его труд. Он работает медленно, так как ему надо просчитать множество значений столбиком на листочке. Если бухгалтер вовремя не заполнит и не отправит отчёт в контролирующий орган, тот выставит штраф организации за несвоевременное предоставление отчётов, и организация понесёт финансовые потери.

Ещё пример. Разрабатывая калькулятор, допустили ошибку и вместо знака «плюс» на кнопке отображается буква «П». Бухгалтер, немного изучив программу, понял: по этой кнопке происходит сложение чисел, вовремя заполнил отчёты и сдал их в контролирующий органы. Во время работы он чувствовал неудобство, поскольку приходилось постоянно помнить, что означает буква «П» на кнопке калькулятора, но это не оказало критичного влияния на его работу и на организацию в целом.

Мы видим, что дефекты оказывают разное влияние на программу и пользователей. Одни дефекты незначительны и не доставляют особых проблем, а другие полностью блокируют работу. Всё зависит от степени критичности дефекта – её ещё называют «серьёзностью дефекта».

Критичность – важность воздействия конкретного дефекта на функционирование и возможность использования программы.

У дефектов есть пять уровней критичности:
– блокирующий;
– критический;
– значительный;
– незначительный;
– тривиальный.

Блокирующий – дефект полностью блокирует выполнение функционала программы, и нет никакого способа его обойти.

Рассмотрим блокирующий дефект на примере программы для сложения чисел. В ней есть кнопка «Сложить», однако она заблокирована, и введя числа для сложения мы не можем их никак сложить. У нас нет возможности обойти этот дефект и каким-либо образом запустить функцию сложения в программе.

Критический – дефект блокирует выполнение функционала программы, но есть альтернативный путь для его обхода.

Рассмотрим критический дефект на примере. В программе есть кнопка «Сложить», однако она заблокирована, и введя числа для сложения мы не можем их сложить с помощью данной кнопки. Но у нас есть возможность обойти данный дефект: необходимо после ввода чисел установить курсор мыши на любое поле для ввода числа и нажать кнопку «Enter» на клавиатуре, и программа произведёт сложение.

Значительный – дефект, указывающий на некорректную работу функционала программы. Проявляется не тем, что функция не работает, а тем, что она работает неправильно.

Рассмотрим значительный дефект на том же примере программы для сложения чисел. Кнопка «Сложить» работает: мы вводим числа в поля для ввода и нажимаем на неё. Программа складывает числа и выводит результат сложения, однако при этом программа перестаёт отвечать на действия пользователя на определённое время (зависает). Через минуту программа снова начинает отвечать на действия пользователя. И так происходит после каждого сложения.

Незначительный – дефект, не относящийся к функциональности программы, очевидная проблема пользовательского интерфейса.

Пример незначительного дефекта. У программы есть кнопка, при нажатии на которую в программе очищаются все поля. В требованиях прописано, что кнопка должна именоваться «Очистить». При разработке программы указали некорректную надпись «Пусто» у кнопки. Она продолжает выполнять свою функцию, но название неверное – не соответствует требованиям.

Незначительная критичность указывается у тех дефектов, которые относятся к удобству использования программы или к интерфейсу программы.

Тривиальный – дефект, не затрагивающий функциональность программы, а также оказывающий минимальное влияние на общее качество работы программы.

Обычно тривиальные дефекты – это грамматические ошибки в интерфейсе программы или в сопроводительной документации к программе, а также дефекты сторонних библиотек или сервисов, не относящихся к самой программе. Часто они трудно отличимы от дефектов незначительной критичности.

Пример тривиального дефекта. Есть библиотека[7] стороннего программиста, в которой прописано: если хочешь получить картинку с природой, то запроси картинку, передав библиотеке слово «природа», картинку с небом – «небо», и так далее. Разрабатывая программу сложения чисел мы решили, что будет замечательно, если интерфейс (фон) программы сделать не одноцветным, а использовать какое-либо изображение, к примеру – изображение неба. Мы это реализовали. Теперь, когда мы запускаем программу, она подключается к библиотеке стороннего программиста и запрашивает у неё картинку неба, передавая библиотеке слово «небо». Однако библиотека возвращает картинку с природой, и мы видим оформление программы с фоном природы. Это явная ошибка в сторонней библиотеке, а не нашей программы. Но так как наша программа использует в своей работе эту стороннюю библиотеку, мы заводим дефект для нашего программиста, чтобы он исправил дефект, связанный с внешним оформлением нашей программы.

Для корректного определения критичности дефектов специалисты по тестированию должны в деталях знать функциональность тестируемой программы и особенности её работы. Это один из признаков, который характеризует хороших тестировщиков.

Кроме критичности у дефектов есть приоритет. В процессе разработки программы специалисты по тестированию находят множество дефектов и каждому присваивают критичность.

[7] Программная библиотека – это набор готовых частей программного кода, которые могут использоваться для создания других программ.

Через определённое время набирается несколько десятков дефектов, которые необходимо устранять с помощью правки программного кода программистом. Он готов устранять дефекты, однако не может одновременно исправлять все. В этот момент ответственные специалисты собираются для определения порядка исправления дефектов. То есть занимаются приоритизацией[8]. Она помогает определить, какие дефекты нужно устранить в первую очередь. Когда специалисты приоритизируют дефекты, они присваивают им определённый приоритет.

Приоритет – степень важности, присваиваемая объекту, которая указывает на очерёдность устранения дефекта или очерёдность выполнения задачи.

Обратите внимание: приоритет есть не только у дефектов, но и у задач[9] (заявок на изменение). Приоритеты у дефектов сообщают нам, в какой очерёдности необходимо устранять дефекты. Приоритеты у задач сообщают, в какой очерёдности необходимо выполнять задачи. Ещё важный момент: в отличие от критичности, приоритет есть и у дефектов, и у задач. Критичность же есть только у дефектов.

У приоритета существует три уровня:

– высокий;

– средний;

– низкий.

В ряде компаний могут вводить дополнительные уровни.

Высокий – требуется устранить или выполнить в первую очередь.

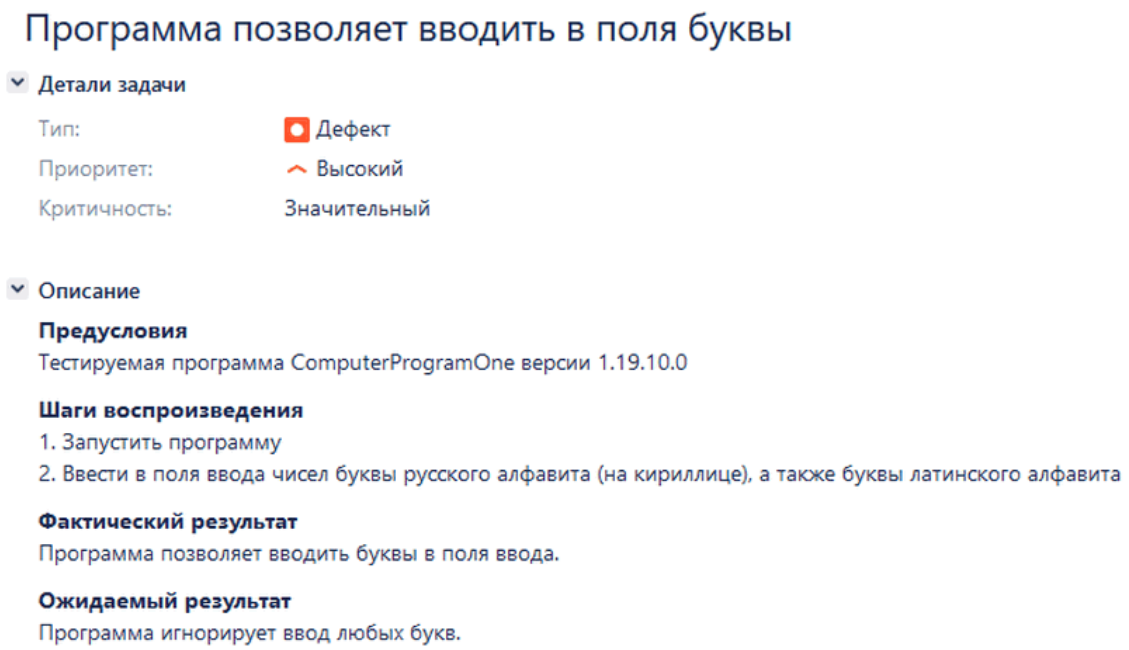
Средний – требуется устранить или выполнить во вторую очередь, когда нет дефектов и задач с высоким приоритетом.

Низкий – требуется устранить или выполнить в последнюю очередь, когда все дефекты и задачи с более высокими приоритетами уже выполнены.

Указание критичности и приоритета является важной частью процесса разработки и тестирования, так как данные атрибуты однозначно классифицируют дефекты по степени их влияния на систему и очерёдность их исправления:

[8] Приоритизация – это процесс определения того, что является важным и должно быть выполнено первым, а что менее важно и может быть выполнено позже.

[9] Задача, заявка на изменение (ЗнИ) – это запрос, который предлагает внести изменения в программу. Он может быть создан как внутри организации, так и внешними пользователями. Он фиксируется и оценивается командой, которая разрабатывает программу. Запрос проходит процесс оценки и утверждения, прежде чем быть включённым в планы на реализацию.



Не во всех организациях используют одновременно два атрибута, чаще всего только приоритет, который логически объединяет оба атрибута, однако это некорректно.

Зачем нужны оба атрибута? Допустим, есть программа, и в ней нашли два дефекта.

Дефект № 1 – не работает функциональность формирования годового отчёта для бухгалтера, которым он должен будет воспользоваться в начале следующего года. Дефект по критичности «блокирующий».

Дефект № 2 – на главной странице сайта есть логотип, в котором имеется опечатка, буквы «к» и «ё» заменили на «т» и «е», и название компании «Клён» читается как «Тлен». Дефект по критичности «тривиальный».

Дефект № 2 может сильно подорвать доверие пользователей к компании и продукции, которую они предлагают, что в свою очередь может повлиять на продажи организации и прибыль. Это зависит от компании и её места на рынке. Дефект № 1 блокирующий, но функционалом не будут пользоваться ещё более полугода, так как сейчас, допустим, лето. В связи с вышесказанным, руководители установят дефекту № 2 высокий приоритет на устранение, а дефекту № 1 средний. Первым будет устраняться дефект № 2.

Приоритеты постоянно пересматриваются, так как они зависят от многих факторов. К примеру, если наступит начало года и отчётный период, а дефект № 1 не позволяет бухгалтерам создавать и отправлять отчёты, то приоритеты у дефектов будут изменены. Дефект № 1 получит высокий приоритет на устранение, а дефект № 2 – средний.

Что такое тестирование

Мы познакомились со специалистами по тестированию и с деятельностью, которой они занимаются. Также изучили, что из себя представляют программы и информационные системы. Настал момент рассмотреть подробнее, что же такое тестирование.

В современном мире программы являются неотъемлемой частью нашей жизни. Они применяются во многих сферах: программы для бизнеса; персональные приложения и игры; программы и микропрограммы в технике (например, телевизоры, стиральные машины). В своей повседневности вы постоянно сталкиваетесь с программами определённого рода.

Мы уже знаем, что программы, работающие некорректно, могут привести к проблемам – потеря времени, денег, деловой репутации, а также могут стать причиной травм и даже смерти. Скорее всего кто-то из вас имел опыт использования программ, которые работали не так, как ожидалось.

Специалисты по тестированию минимизируют риски, выявляя ошибки в программах. Однако это не единственная цель их работы и тестирования в целом. В чём тогда заключается цель тестирования? Она заложена в самом его определении. Рассмотрим ряд существующих неполных и неточных определений тестирования:

– Тестирование ПО – это процесс поиска ошибок в программе. Здесь подразумевается, что мы целенаправленно стараемся найти в программе максимальное количество ошибок.

– Тестирование ПО – это процесс проверки, который гарантирует, что программа работает безошибочно. Здесь имеется в виду, что мы не ищем ошибки, а проводим ряд проверок, чтобы убедиться, что требуемые нам операции отработали без ошибок.

– Тестирование ПО – это процесс, который направлен на подтверждение соответствия программы требованиям, поставленным заказчиком[10]. Здесь подразумевается следующее: мы проверяем, что в программе реализовано то, что попросил заказчик.

Ни одно из этих определений не является полностью верным или точным, так как тестирование – это более сложный и многогранный процесс, который включает в себя множество активностей, методов и подходов. На основании всего вышесказанного выведем определение.

Тестирование ПО – это процесс исследования, испытания программы, с целью продемонстрировать заказчикам и/или заинтересованным лицам, что программа соответствует установленным требованиям, а также это процесс проверки соответствия между реальным поведением программы и ожидаемым поведением, и выявление ситуаций, в которых поведение программы является неправильным или нежелательным.

Данное определение даёт полное понимание, что из себя представляет тестирование программного обеспечения. И ничего сверх того, что сказано в определении, не добавить.

[10] Заказчик – лицо, заинтересованное в разработке программы для своих нужд или для дальнейшей продажи программы пользователям. Заказчиками могут быть физические или юридические лица.

Цели тестирования

Мы уже знаем и понимает, что такое тестирование, кто его проводит и что тестируется. Но с какой целью всё это осуществляется? Чтобы ответить на этот вопрос, необходимо рассмотреть цели тестирования программного обеспечения.

Цель – это то, ради чего осуществляется какой-либо процесс. Правильно определённая цель позволяет эффективно организовать работу и направить усилия всех участников на достижение желаемого результата. Перечислим цели тестирования, которые применимы к любой разработке.

Цель № 1. Оценка состояния требований, пользовательских историй, проектной документации или написанного кода для выявления расхождений с первоначально запланированными результатами и формирования предложений по их улучшению. Специалисты проводят детальный анализ всех имеющихся артефактов проектирования и разработки программного обеспечения – требований, пользовательских историй, технического задания, дизайн-документации, написанного кода. Цель данного анализа – проверить их корректность и соответствие первоначальному замыслу и плану. Специалисты ищут расхождения между тем, что было описано или запланировано на этапе проектирования, и тем, как это выглядит на данный момент на этапе разработки. Выявленные расхождения анализируются и на их основе разрабатываются конкретные предложения по улучшению.

Цель № 2. Проверка выполнения всех требований, т. е. проверка, что разработанное программное обеспечение полностью отвечает всем изначально заявленным пользовательским нуждам и ожиданиям. При разработке программного обеспечения сначала определяются все необходимые функции и возможности, которыми должна обладать программа. Это фиксируется в виде требований к программе. Затем приступают к непосредственной разработке самого программного кода. Программисты стараются реализовать все заявленные в требованиях функции. Однако чтобы убедиться, что разработанная программа действительно соответствует изначальным требованиям, проводится его тестирование. В ходе тестирования проверяется, был ли воплощён в жизнь каждый пункт требований. То есть проверяется, реализован ли в программном коде весь функционал, который был описан на этапе определения требований.

Цель № 3. Проверка, что объект тестирования завершён и работает, как ожидают пользователи и заинтересованные лица. В этом случае подтверждаем: программа соответствует ожиданиям и потребностям пользователей, которые будут взаимодействовать с этой программой. Специалисты по тестированию проверяют, работает ли программа так, как этого ожидают пользователи при выполнении тех или иных действий. Проверяется, насколько удобным и понятным является интерфейс программы для пользователей, насколько быстро и эффективно пользователи смогут выполнять свои задачи с помощью этой программы. Также проверяется, учтены ли потребности и ожидания других заинтересованных сторон, таких как администраторы, менеджеры, техподдержка и другие сотрудники организации, в которой будет использоваться данная программа. В результате проверки подтверждается, что разработанная программа полностью соответствует целям её создания и будет удовлетворять нужды всех целевых групп пользователей.

Цель № 4. Создание уверенности в уровне качества объекта тестирования. Проверяется программа, чтобы убедиться, что она соответствует предъявляемым к ней критериям и параметрам качества. Есть, к примеру, зафиксированные параметры качества программы: она должна запускаться и отображать свой интерфейс через 5 секунд после запуска; документ в программе должен сохраняться за 2 секунды; в программе не должно быть критических дефектов и т. д. Специалисты проверяют, что программа соответствует установленным параметрам качества.

Цель № 5. Предотвращение дефектов, т. е. максимальное сокращение количества дефектов путём их предотвращения на ранних этапах создания программного обеспечения. Предотвращение дефектов на этапе сбора и анализа требований. На этом этапе тестировщик проверяет требования на корректность, непротиворечивость и полноту описания. При обнаружении несоответствий или неясностей в требованиях они уточняются ещё на этапе сбора, до начала разработки, что позволяет избежать ошибок на последующих этапах. Предотвращение дефектов на этапе проектирования. На этом этапе специалист по тестированию проверяет архитектурные решения, дизайн и проектирование программы на соответствие требованиям. Если обнаруживаются расхождения, они устраняются специалистами по проектированию на этапе проектирования, до начала реализации. Предотвращение дефектов на этапе разработки. Здесь специалист по тестированию проверяет программу на соответствие требованиям ещё до встраивания программы в общую экосистему. Программисты, в свою очередь, проверяют исходный программный код на корректность и соответствие их стандартам. Это позволяет выявлять и устранять ошибки на самых ранних этапах жизненного цикла программы.

Цель № 6. Обнаружение отказов и дефектов в ПО. В этом случае мы выявляем уже занесённые в результате разработки ПО дефекты. Это могут быть: дефекты в логике работы программы; ошибки в алгоритмах; дефекты в интерфейсе и взаимодействии с пользователем; дефекты в работе с внешними системами и базами данных; недочёты в документации и описании функционала. Проводится систематический поиск и выявление таких дефектов. Это позволяет устранить их до выхода в продуктивное окружение, чтобы обеспечить его корректную и безаварийную работу. В идеале, все дефекты должны быть найдены и исправлены на этапе тестирования, прежде чем программа будет поставлена пользователю.

Цель № 7. Предоставление заинтересованным лицам достаточной информации, позволяющей им принять обоснованные решения в отношении уровня качества объекта тестирования. В ходе тестирования специалисты по тестированию проверяют соответствие программы требованиям, выявляют дефекты. На основании полученных результатов формируют отчёты, в которых указывают количество и критичность найденных дефектов, степень реализации функциональных и нефункциональных требований и другие показатели качества. Эта информация позволяет заинтересованным сторонам принять обоснованное решение о готовности программы: нужна ли доработка для устранения дефектов, возможен ли запуск в эксплуатацию или требуется ещё один цикл разработки и тестирования.

Цель № 8. Соблюдение договорных, правовых или нормативных требований, или стандартов и/или проверка соответствия объекта тестирования таким требованиям и стандартам. Специалисты по тестированию проверяют, что разрабатываемая программа соответствует государственным или международным стандартам, законодательству и т. д. Пример: программа работает с платёжными системами и должна соответствовать их стандартам и требованиям. При тестировании специалисты должны убедиться в этом, в противном случае данная программа не будет допущена к работе с платёжными системами.

Цели тестирования могут отличаться, в зависимости от этапа жизненного цикла программы, на котором проводится тестирование, а также в зависимости от назначения и типа тестируемого компонента или программы.

Верификация и валидация

Проводя тестирование, мы в этот момент проводим верификацию и валидацию программы. Кого-то смутят уже сами эти слова и явно не появится желания погружаться в определения непонятных терминов. Однако, придётся в них разобраться, так как специалисты по тестированию сталкиваются с ними постоянно.

Верификация – подтверждение того, что заданные требования полностью реализованы в программе.

Подтвердить, что заданные требования полностью реализованы в программе – означает необходимость убедиться, что программисты сделали то, что заказчик зафиксировал в требованиях. Рассмотрим на примере. У заказчика есть требование к программе по сложению чисел. Смотрите таблицу:

Номер	Название	Требование
ФТ-1.4	Поле ввода первого числа	В поле можно вводить только положительные целые числа. Максимальное количество вводимых и отображаемых символов 4 (четыре). В поле можно вводить информацию с клавиатуры. В поле можно вставлять данные из буфера обмена. Из поля можно копировать данные в буфер обмена.

Верифицируя программу при проведении тестирования, мы должны проверить, что изложенные требования реализованы и всё сделано так, как требовалось. Мы проверяем, что можно вводить только положительные целые числа, что максимальное количество вводимых и отображаемых символов равно четырём, что в поле можно вводить информацию с клавиатуры, вставлять данные из буфера обмена и из поля можно копировать данные в буфер обмена. Если программа одну из проверок не пройдёт, к примеру, из поля нельзя будет копировать данные, то программа не она верификацию, так как не все требования реализованы. Если же все требования соблюдены, программа пройдёт верификацию. С этим разобрались.

Теперь рассмотрим валидацию.

Валидация – подтверждение того, что функции программы при её использовании соответствуют требованиям и ожиданиям заказчика и программа способна выполнять задачи, которые от неё ожидают.

Валидируя программу, мы должны проверить, что реализованная программистами функциональность соответствует требованиям и ожиданиям заказчика. Рассмотрим на примере. У заказчика есть требование к программе по сложению чисел. Смотрите таблицу:

Номер	Название	Требование
ФТ-1.6	Поле отображения результатов	В поле отображаются результаты сложения чисел полей «ввода первого числа» и «ввода второго числа». Поле заблокировано от ввода информации с клавиатуры и от вставки из буфера обмена. Из поля можно копировать данные в буфер обмена.

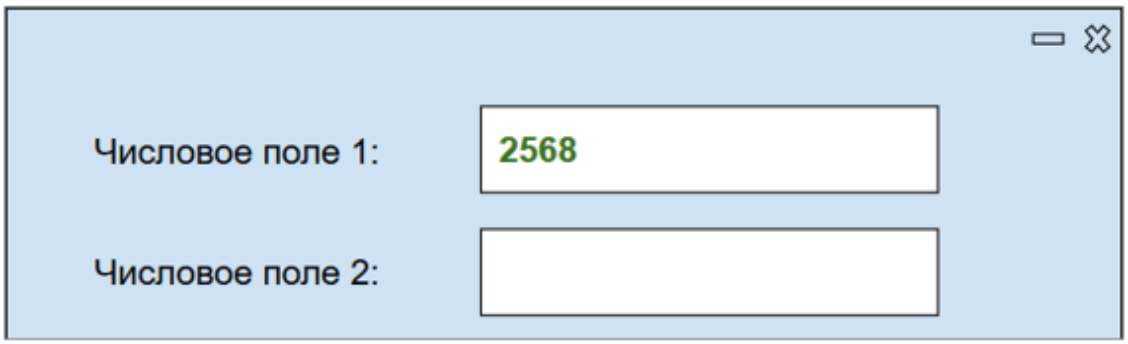
Проводя верификацию, мы убедились, что в поле отображения результатов после сложения чисел отображается результат. Однако через 10 секунд он исчезает. В требованиях не сказано, сколько времени отображать результат сложения, поэтому верификация пройдена. Однако заказчики ожидают, что он не будет исчезать. И логически мы это понимаем. Т. е. программа соответствует требованиям, но не соответствует ожиданиям заказчика, а это означает, что она не прошла валидацию. Если все требования соблюдены, и программа работает, как ожидает заказчик, она пройдёт валидацию.

Позитивное и негативное тестирование

В программе есть заявленный функционал, который описан в требованиях, и, проверяя данный функционал, мы проводим позитивное тестирование, т. е. убеждаемся, что программа работает так, как описано в требованиях, при использовании допустимых и корректных данных. Нам также необходимо проверить, как программа будет работать, если использовать некорректные данные и непредусмотренные ситуации – это уже негативное тестирование.

Позитивное тестирование – тестирование, которое определяет, что программа работает так, как ожидалось, и использует для проверок только корректные данные.

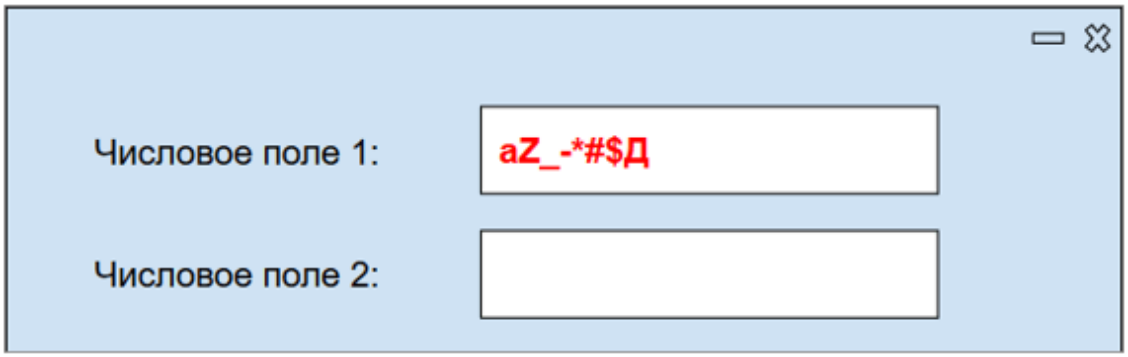
Пример позитивного теста. В программе есть поле ввода, которое может принимать только цифры. При тестировании мы проверяем, что в поле можно вводить цифры. Ввод других данных не проверяется:



Если в процессе позитивного тестирования мы не сможем ввести цифры в поле ввода, тест не пройден. Если сможем – тест пройден.

Негативное тестирование – тестирование, которое гарантирует, что программа может корректно обрабатывать неверный ввод данных или неожиданное поведение пользователя.

Пример негативного теста. В программе есть поле ввода, способное принимать только цифры. При тестировании мы проверяем, что в поле нельзя ввести буквы, спецсимволы, знаки препинания. Программа при этом должна сообщать нам о невозможности ввода некорректных данных, а также не должна аварийно завершать работу:



Если в процессе негативного тестирования программа позволит ввести вместо цифр буквы и попытаться их сложить, а ещё хуже – при этом аварийно завершит свою работу значит тест не пройден. Если программа не позволит вводить ничего кроме цифр или даст вводить буквы, но не позволит запустить функцию сложения, тест пройден.

Ложноположительные и ложноотрицательные результаты тестирования

В процессе проведения тестирования специалисты по тестированию могут принять верное поведение программы за дефект, а неверное поведение – за норму. В этом случае говорят, что получены ложноотрицательный или ложноположительный результаты тестирования. Иногда их называют «ложнонегативными» и «ложнопозитивными» результатами тестирования.

Ложноположительные результаты тестирования – во время проверки функционала программы принимаем неправильное поведение программы за её корректное поведение.

Ложноположительные результаты тестирования возникают в следующей ситуации. Представим, что мы проверяем работу некой функции программы. Ожидается, что при определённых входных данных функция должна возвращать конкретный результат. Однако в тесте мы подаём на вход другие данные, при которых поведение функции не определено требованиями. И если на эти непредвиденные данные функция вернула какой-то результат, мы по ошибке принимаем такое поведение за правильное. Хотя на самом деле функция работает неправильно, выдавая случайный результат вместо сообщения об ошибке. То есть мы ошибочно интерпретируем неправильную работу программы как корректную. Из-за этого подобные дефекты остаются незамеченными во время тестирования.

Ложноотрицательные результаты тестирования – во время проверки функционала программы, принимаем правильное поведение программы за её некорректное поведение.

Ложноотрицательные результаты тестирования возникают в следующей ситуации. Допустим, мы проверяем работу функции на некотором входном значении. Спецификация гласит, что на этих данных функция должна вернуть определённый результат. Однако в тесте функция вернула другое значение, отличное от ожидаемого. На первый взгляд это выглядит как дефект в работе функции. На самом деле возвращаемое значение также является корректным согласно дополнительным неформальным требованиям, о которых специалист по тестированию не знал. То есть функция работает правильно, но из-за неполноты требований тестировщик ошибочно интерпретирует это как дефект. Таким образом из-за ложноотрицательного результата корректно работающий функционал маркируется как дефектный. Это приводит к необоснованной потере времени на поиск несуществующих дефектов.

Ручное и автоматизированное тестирование

Тестирование может проводиться как вручную, так и автоматизировано с помощью специализированных инструментов, которые позволяют выполнять тесты не вручную, а автоматизировано. Рассмотрим эти понятия.

Ручное тестирование – процесс тестирования программного обеспечения вручную без использования программных средств, которые выполняют проверки функционала программы с помощью автоматизированных сценариев тестирования.

Пример: запускаем программу и начинаем её тестировать, вручную нажимать на кнопки, вводить данные и смотреть на получаемые результаты. Если вы сейчас самостоятельно начнёте проверять корректность работы любой программы, вы как раз будете проводить ручное тестирование.

В его определении мы упомянули автоматизированный сценарий тестирования, который также именуют автоматизированным тестом.

Автоматизированный сценарий тестирования – это набор действий, описанный на определённом языке программирования, которые выполняются автоматически с использованием специальных инструментов или программного обеспечения для проверки определённой функции тестируемой программы.

Их разрабатывают специалисты по автоматизированному тестированию. Такие сценарии тестирования позволяют проверить работоспособность и качество программы, автоматизируя процесс. Автоматизированные сценарии тестирования могут включать в себя различные действия, такие как ввод данных, выполнение определённых операций, проверку результатов и сравнение их с ожидаемыми значениями. Они помогают ускорить и упростить процесс тестирования, а также повысить его надёжность и точность, и являются основой автоматизированного тестирования.

Автоматизированное тестирование – процесс тестирования программного обеспечения с использованием программных средств для выполнения автоматизированных сценариев тестирования и проверки результатов выполнения, с целью сокращения времени тестирования и упрощения процесса.

Чтобы заниматься таким тестированием необходимо обладать специализированными навыками и знать языки программирования.

Какое тестирование можно автоматизировать?

– функциональное тестирование сайтов;
– функциональное тестирование настольных приложений;
– функциональное тестирование мобильных приложений;
– тестирование API информационных систем.

Есть и другие сферы применения автоматизированного тестирования. Некоторые ключевые характеристики автоматизированного тестирования:

– Можно запускать тесты регулярно и независимо от занятости человека. Это обеспечивает непрерывное тестирование.
– Ускоряет и упрощает процесс тестирования.

– Повышает надёжность тестирования благодаря исключению из процесса человеческого фактора.

К примеру, необходимо ежедневно проверять, что после выпуска очередной новой версии программы в ней не ломаются основные функции, которые до этого работали без ошибок. Для этого требуется каждый день выполнять сотни тестов – однообразная и не воодушевляющая специалистов деятельность. В таком случае при наличии навыков и опыта специалисты разрабатывают сотни автоматизированных сценариев тестирования, которые в дальнейшем будут автоматически запускаться ежедневно, проверять программу, и заинтересованным людям будут отправляться на почту отчёты с результатами тестирования. Специалисты в это время могут посвятить себя более интеллектуальному тестированию.

Тестовое окружение и не только

В процессе своей трудовой деятельности специалисты по тестированию работают с окружениями. Тестовое окружение и продуктивное окружение – это два основных типа окружений при разработке программного обеспечения. Под окружением в данном случае понимается функционально связанный между собой набор компьютеров, программ, информационных систем.

Продуктивное окружение – это окружение, в котором запускается готовое программное обеспечение для конечных пользователей.

Когда вы посещаете интернет и заходите на сайт поисковой системы, чтобы осуществить поиск какой-либо информации, вы попадаете на сайт, работающий в промышленном окружении, куда поместили его создатели, чтобы конечные пользователи им пользовались.

Основные характеристики продуктивного окружения:

– в нём работает окончательная версия программы после всех тестов;

– в нём не разрабатывается и не тестируется программа;

– оно не используется для исправления дефектов;

– обеспечивает максимальную производительность и доступность;

– содержит реальные данные и учётные записи пользователей;

– настроена на безопасность и конфиденциальность данных;

– требует высокой доступности и производительности по сравнению с тестовым окружением.

Тестовое окружение – окружение, предназначенное для тестирования и отладки программ.

Перед тем как поместить сайт в промышленное окружение создатели помещают его в тестовое окружение, которое не доступно для конечных пользователей. В тестовом окружении проводят тестирование поискового сайта, чтобы убедиться, что сайт работает и его можно в дальнейшем поместить в промышленное окружение.

Основные характеристики тестового окружения:

– в нём разрабатывается и тестируется программа до выпуска в продуктивное окружение;

– используется для исправления дефектов и проверки функциональности до переноса в промышленное окружение;

– имитирует реальные условия работы, но может иметь меньшие мощности по сравнению с промышленным окружением;

– содержит не реальные данные, а тестовые.

Таким образом, тестовое окружение – для разработки и тестирования, продуктивное окружение – для реальной эксплуатации готовой программы. Основная работа специалистов по тестированию проводится на тестовом окружении.

В различных компаниях могут использоваться и другие окружения. Примеры названий: dev (дев), test (тест), qa (кью-эй), preprod (препрод), stage (стейдж), preview (превью) и т. д. То, какие окружения используются, зависит от специфики работы организации и разрабатываемых программ, поэтому мы не будем их рассматривать. Но вы должны знать, что кроме продуктивного и тестового окружения в ряде организаций существуют и другие окружения.

Тестовая документация и артефакты

В процессе работы специалисты по тестированию создают различные артефакты[11] и документы. Кстати, дефект – это один из артефактов работы тестировщика. Познакомимся поближе с документами и артефактами, которые могут создавать и с которыми могут работать специалисты по тестированию.

[11] Артефакт – это результат какой-либо деятельности. Им может быть любая документация, написанный код и прочее.

План тестирования

Тестирование, как и любой другой процесс, должно планироваться. Планирование тестирования – это одна из активностей, о которой мы поговорим в других главах книги. Планирование является непрерывной деятельностью, которая выполняется в течение всего жизненного цикла ПО, и в процессе планирования создаётся план тестирования (тест-план).

План тестирования – документ, описывающий стратегию и тактику тестирования программного обеспечения.

Это документ, который помогает организовать и планировать процесс тестирования на этапе разработки программного обеспечения, что позволяет провести тестирование эффективно и качественно.

В плане тестирования фиксируется следующая информация:

– Цели и задачи тестирования. Здесь определяется, что именно нужно проверить в программном обеспечении.

– Объекты тестирования[12]. Какие модули, функциональные возможности, интерфейсы и т. д. будут тестироваться.

– Уровни, типы и виды тестов. Например, функциональное тестирование, нефункциональное тестирование, тестирование производительности, и т. д.

– Приоритеты тестов. В какой последовательности будут выполняться тесты.

– Ответственные за тестирование. Кто конкретно будет разрабатывать, выполнять и отслеживать результаты тестов.

– График тестирования. План со сроками, в котором указано, когда и в какие сроки должно быть проведено и завершено тестирование.

– Тестовое окружение. Какое тестовое окружение будет использоваться в тестах.

– Риски. Что может увеличить сроки тестирования или блокировать тестирование и как эти риски нивелировать.

– Критерии успешности тестирования. Что считать успешным тестированием и какие условия должны выполняться.

В зависимости от организации в план тестирования могут включать и другую информацию, которую посчитают важной. С примером плана тестирования вы можете ознакомиться на сайте автора[13].

Разработку планов тестирования проводят опытные специалисты по тестированию или руководители и менеджеры по тестированию.

[12] Объект тестирования – компонент или программа, которые должны быть протестированы.

[13] Ознакомиться с планом тестирования можно по ссылке https://victorz.ru/books/book-1

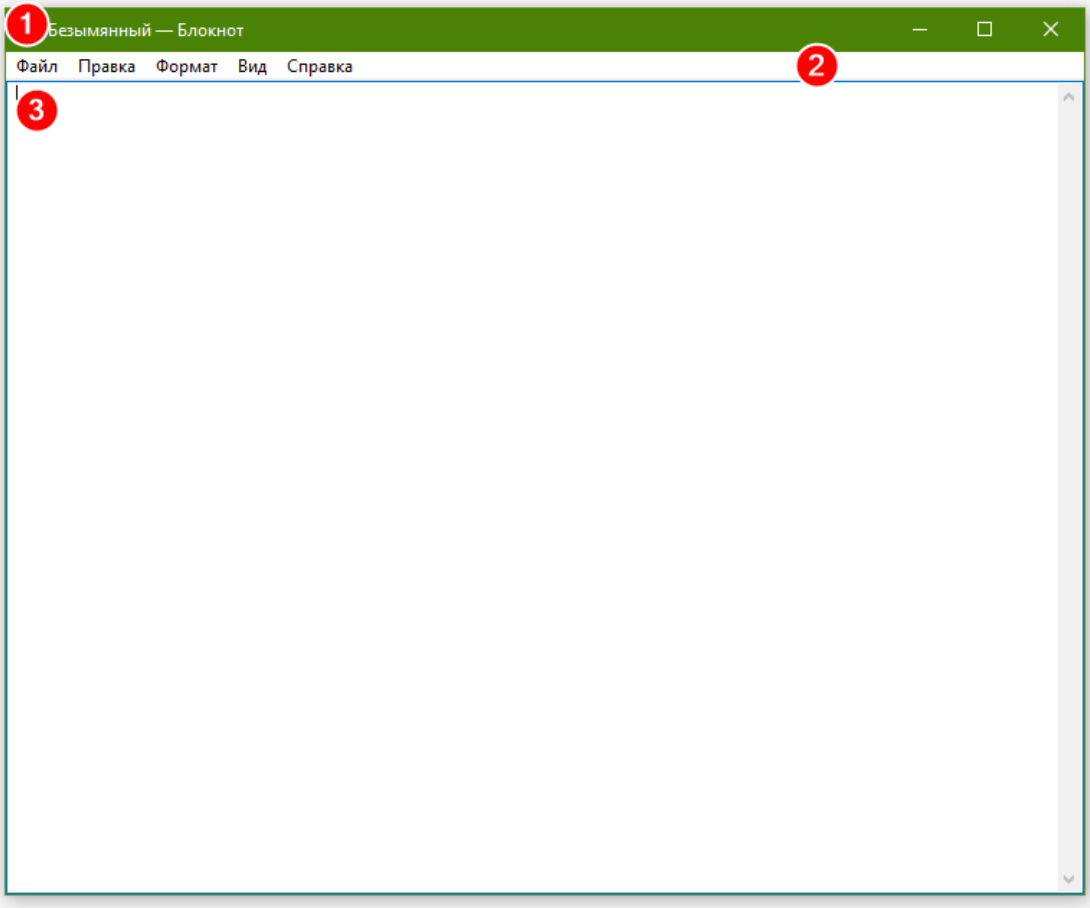
Функциональная карта

Функциональную карту ещё называют «интеллектуальной картой», «диаграммой связей», «ассоциативной картой» и т. д. Мы её называем «функциональной», так как на ней отражены функциональные области программы, которые необходимо тестировать.

Функциональная карта программы – схема, которая визуально отображает основные функциональные возможности программы и позволяет быстро понять её назначение и принцип работы.

Цель функциональной карты – дать общее представление о структуре и логике работы программы, выделить её основные возможности. Она используется на этапе планирования и проектирования тестов. В функциональной карте отображают основные модули или разделы программы, вспомогательные функции, обеспечивающие работу основных модулей.

Как функциональные карты применяются специалистами по тестированию в работе? Создавая функциональную карту программы, специалист разбивает программу на логические функциональные блоки и описывает её ветвлениями. Рассмотрим на простом примере, взяв за основу программу «Блокнот», которая имеется в операционной системе Windows:



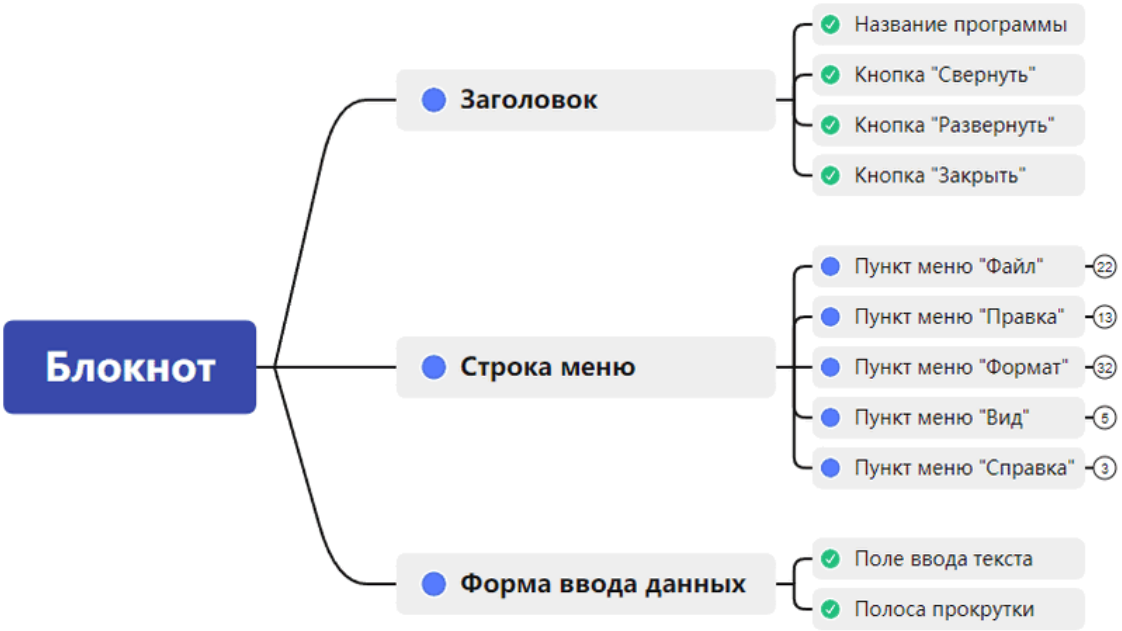
У программы «Блокнот» есть «Заголовок» (1), «Строка меню» (2), «Форма ввода данных» (3). Названные блоки (элементы) в свою очередь делятся на дополнительные элементы.

Заголовок (1) имеет:

– название программы в заголовке;

– кнопка «Свернуть»;

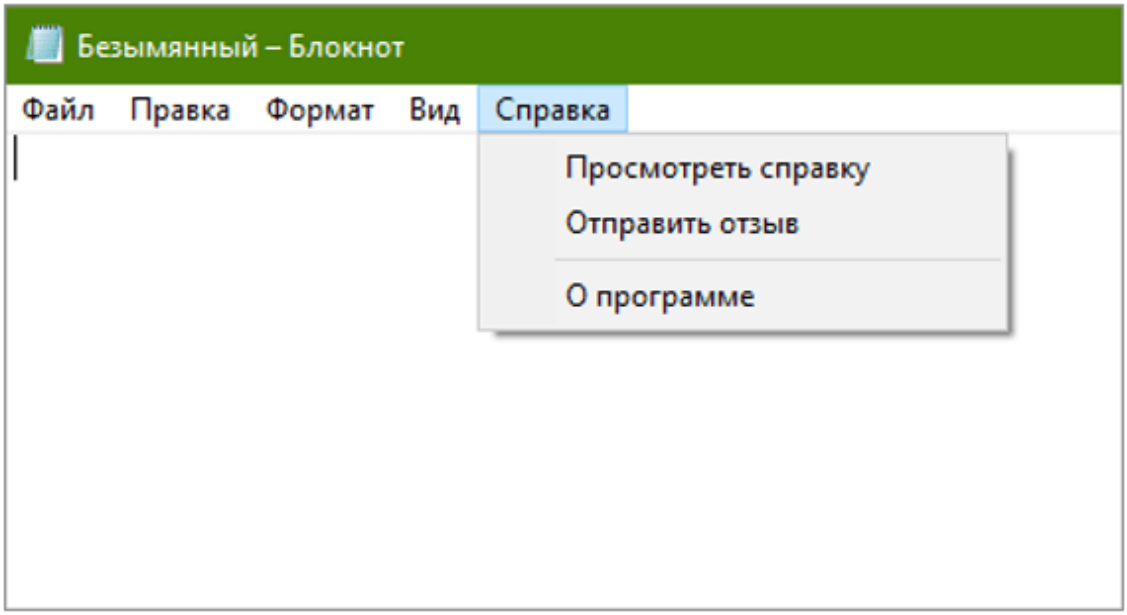
– кнопка «Развернуть»;
– кнопка «Закрыть».
Строка меню (2) имеет:
– пункт меню «Файл»;
– пункт меню «Правка»;
– пункт меню «Формат»;
– пункт меню «Вид»;
– пункт меню «Справка».
Форма ввода данных (3) имеет:
– поле ввода текста;
– полоса прокрутки.
Всё перечисленное можно отобразить на функциональной карте:



На функциональной карте видим, как ветвится функционал программы. Обратите внимание, что для удобства восприятия различные пункты помечаем двумя видами значков, которые могут быть другими:

– синий круг – это значит ветвление будет дальше продолжаться;
– зелёный круг с галочкой – это является конечной проверкой.

Визуально видим, как ветвится функционал программы и остаётся только продолжить следование по пунктам меню или функционалу программы. Для большей наглядности продолжим разбор пункта меню «Справка»:



Пункт меню «Справка» имеет пункты:
– просмотреть справку.
– отправить отзыв.
– о программе.
Функциональная карта получит следующее продолжение:

Таким образом следуя по всей программе, специалист описывает её функциональной картой. Если он где-то оставил значок синего круга и начал описывать другие ветки, то точно не забудет вернуться, чтобы продолжить описывать функциональную ветку программы, пока не доберётся до конечных пунктов проверки.

После того как тестировщик полностью создаст функциональную карту, он будет видеть все проверки, которые необходимо провести в программе.

Тест-кейс

Тест-кейс – набор входных значений, предусловий выполнения, ожидаемых результатов и постусловий выполнения, разработанный для определённой цели или тестового условия, таких как выполнения определённого пути программы или же для проверки соответствия определённому требованию.

Предполагаю, что, прочитав это официальное описание, вы загрустили, ничего не поняв. Попробую сформулировать определение, используя понятные для всех формулировки.

Тест-кейс – это чёткое описание действий, которые необходимо выполнить, чтобы проверить работу программы (поля для ввода, кнопки и т. д.). Данное описание содержит: действия, которые надо выполнить до начала проверки – предусловия; действия, которые надо выполнить для проверки – шаги проверки; описание того, что должно произойти, после выполнения действий для проверки – ожидаемый результат; действия, которые необходимо выполнить в самом конце, чтобы привести систему в первоначальное состояние, сбросив все внесённые нами изменения – постусловия.

Второе определение понятнее, однако всё равно необходимо напрячься, чтобы понять написанное, поэтому следующее определение будет ещё проще и дано на языке простого обывателя.

Тест-кейс – это описание того, что надо сделать, чтобы проверить определённый функционал программы и что должно произойти после того, как мы выполним описанные действия.

Почему здесь приведено так много определений? Для того, чтобы вы чётко осознали, что такое тест-кейс, так как в своей работе специалист по тестированию очень много времени уделяет работе с тест-кейсами: написание, правка, проверка программ по тест-кейсам и т. д.

Тест-кейсы также называют «контрольными примерами» или «сценариями тестирования». В определениях терминов в данной книге вы с этим названием будете сталкиваться.

Рассмотрим основные атрибуты, из которых состоит тест-кейс и которые используются в большинстве организаций:

1) Номер тест-кейса – уникальный идентификатор тест-кейса. Если у вас тысячи тест-кейсов, то при общении с коллегами вам будет проще сообщить номер тест-кейса, ссылаясь на него, а не пытаться словами рассказать, где и как найти определённый тест-кейс.

2) Заголовок – краткое, понятное и ёмкое описание сути проверки.

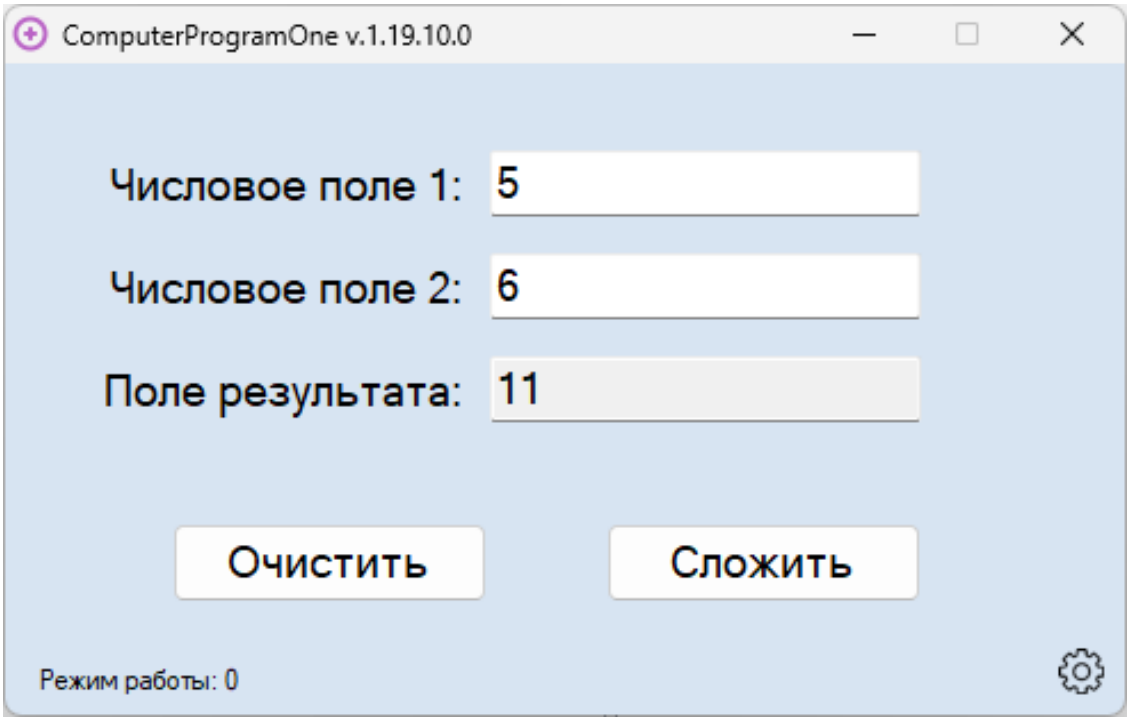
3) Предусловия – описание действий, которые необходимо предварительно выполнить или учесть перед началом проведения проверок.

4) Шаги проверки – описание последовательности действий, которые надо выполнить для проверки.

5) Ожидаемый результат – проверка, которая устанавливает, что мы ожидаем получить после выполнения определённых действий в соответствующем шаге.

В зависимости от специфики организации могут присутствовать дополнительные атрибуты для заполнения: постусловия, приоритет, функциональный блок, программа, ссылка на требование, номер требования и т. д.

Рассмотрим тест-кейс на примере программы для сложения чисел:



Для тестирования программы необходимо подготовить список проверок и написать тест-кейсы. Одной из проверок будет сложение однозначных положительных целых чисел. Специалисту по тестированию надо на эту проверку написать тест-кейс. Предположим, что это будет двенадцатый тест-кейс, который он начал писать на данную программу:

	Номер тест-кейса:	12
	Название:	Сложение однозначных положительных целых чисел
	Предусловия:	Программа «Computer Program One» запущена
	Шаги	**Ожидаемый результат**
1	Ввести с клавиатуры в поле «Числовое поле 1» любое однозначное положительное целое число.	В поле «Числовое поле 1» отображается введённое число.
2	Ввести с клавиатуры в поле «Числовое поле 2» любое однозначное положительное целое число.	В поле «Числовое поле 2» отображается введённое число.
3	Нажать на кнопку «Сложить».	В поле «Поле результата» отображается сумма чисел «Числовое поле 1» + «Числовое поле 2».

Видим готовый тест-кейс. Используя его, уже можно провести одну проверку программы на сложение положительных целых чисел. Суть понятна. Теперь рассмотрим правила написания тест-кейсов. Их не нужно заучивать. При необходимости можете открыть данную книгу и повторить правила перед тем, как начать писать тест-кейсы.

Правило № 1. Заголовок:

– должен быть чётким, кратким, понятным и однозначно характеризующим суть тест-кейса;

– не может содержать выполняемые шаги и ожидаемый результат.

Правило № 2. Предусловие:

– может содержать полную информацию о состоянии системы или объекта, необходимом для начала выполнения шагов тест-кейса;

Конец ознакомительного фрагмента.