Методические указания по выполнению лабораторных работ по дисциплине «Тестирование программного обеспечения» для студентов технических специальностей и слушателей факультета дополнительного образования

ВВЕДЕНИЕ

Учебная дисциплина «Тестирование программного обеспечения» является составной частью цикла дисциплин по информационным технологиям, изучаемых студентами специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)». Занимая важное место в общепрофессиональной подготовке студентов, учебная дисциплина «Тестирование программного обеспечения» обеспечивает подготовку специалиста, владеющего фундаментальными знаниями и практическими навыками в области тестирования программного обеспечения.

В методическом пособии представлены материалы авторитетных специалистов в области тестирования, как нашего времени, так и классиков прошлых лет. Лабораторные занятия предназначены для закрепления и более глубокого изучения определенных аспектов лекционного материала на практике.

Материалы к лабораторным работам располагаются в Интернете по адресу https://github.com/tsarik-polessu/software-testing

1. ОСНОВНЫЕ ПОНЯТИЯ ТЕСТИРОВАНИЯ

Тестирование программного обеспечения – процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта.

Тестирование – один из наиболее трудоемких этапов создания программ. Трудоемкость этапа тестирования составляет от 30% до 60% общей трудоемкости отводимой на разработку программного обеспечения. Доля стоимости этапа тестирования в общей стоимости программ имеет тенденцию возрастать при увеличении сложности комплексов программ и повышении требований к их качеству.

1.1. Эволюция понятия тестирования

Можно выделить несколько основных «эпох тестирования».

В 50-60-х годах прошлого века процесс тестирования был предельно формализован, отделен от процесса непосредственной разработки программного обеспечения и «математизирован». Фактически тестирование представляло собой отладку программ (debugging). Существовала концепция «исчерпывающего тестирования», т.е. проверки всех возможных путей выполнения кода со всеми возможными входными данными. Но очень скоро было выяснено, что исчерпывающее тестирование невозможно, т.к. количество возможных путей и входных данных очень велико, и при таком подходе сложно найти проблемы в документации.

В 70-х годах фактически родились две фундаментальные идеи тестирования. Тестирование сначала рассматривалось как процесс доказательства работоспособности программы в некоторых заданных условиях (positive testing), а затем – строго наоборот, как процесс доказательства неработоспособности программы в некоторых заданных условиях (negative testing). В наши дни эти идеи отмечаются как две взаимодополняющие цели тестирования.

В 80-х годах произошло ключевое изменение места тестирования в разработке программного обеспечения. Вместо одной из финальных стадий создания проекта, тестирование стало применяться на протяжении всего цикла разработки (software lifecycle), что позволило не только быстро обнаруживать и устранять проблемы, но даже предсказывать и предотвращать их появление.

В 90-х годах произошел переход от тестирования как такового к процессу, который называется «обеспечение качества» (quality assurance), который охватывает весь цикл разработки программного обеспечения и затрагивает процессы планирования, проектирования, создания и выполнения тест-кейсов, поддержку имеющихся тест-кейсов и тестовых окружений. Тестирование вышло на качественно новый уровень, который естественным образом привел к дальнейшему развитию методологий, появлению достаточно мощных инструментов управления процессом тестирования и инструментальных средств автоматизации тестирования, уже вполне похожих на своих нынешних потомков.

В нулевые годы нынешнего века развитие тестирования продолжалось в контексте поиска все новых и новых путей, методологий, техник и подходов к обеспечению качества. Серьезное влияние на понимание тестирования оказало появление гибких методологий разработки и таких подходов, как «разработка под управлением тестированием» (test-driven development, TDD).

Автоматизация тестирования уже воспринималась как обычная неотъемлемая часть большинства проектов, а также стали популярны идеи о том, что во главу процесса тестирования следует ставить не соответствие программы требованиям, а ее способность предоставить конечному пользователю возможность эффективно решать свои задачи.

Современный этап развития тестирования характеризуется следующими основными характеристиками:

- гибкие методологии и гибкое тестирование;
- глубокая интеграция с процессом разработки;
- широкое использование автоматизации, колоссальный набор технологий и инструментальных средств;
- кроссфункциональность команды, когда тестировщик и программист во многом могут выполнять работу друг друга.

1.2. Жизненный цикл разработки программного обеспечения

Рассмотрим модели разработки программного обеспечения как часть жизненного цикла программного обеспечения. Модель разработки программного обеспечения – структура, систематизирующая различные виды проектной деятельности, их взаимодействие и последовательность в процессе разработки программного обеспечения. Выбор той или иной модели зависит от масштаба и сложности проекта, предметной области, доступных ресурсов и множества других факторов.

Выбор модели разработки программного обеспечения серьезно влияет на процесс тестирования, определяя выбор стратегии, расписание, необходимые ресурсы и т.д.

Рассмотрим следующие модели разработки программного обеспечения: водопадную, итерационную инкрементальную и гибкую.

Водопадная модель

Водопадная модель (waterfall model) сейчас представляет скорее исторический интерес, т.к. в современных проектах практически неприменима. Она предполагает однократное выполнение каждой из фаз проекта, которые, в свою очередь, строго следуют друг за другом. Очень упрощенно можно сказать, что в рамках этой модели в любой момент времени команде «видна» лишь предыдущая и следующая фазы. В реальной же разработке программного обеспечения приходится «видеть весь проект целиком» и возвращаться к предыдущим фазам, чтобы исправить недоработки или что-то уточнить.

К недостаткам водопадной модели принято относить тот факт, что участие конечных пользователей в ней либо не предусмотрено вообще, либо предусмотрено лишь косвенно на стадии однократного сбора требований. С точки зрения же тестирования эта модель плоха тем, что тестирование в явном виде появляется здесь лишь с середины развития проекта, достигая своего максимума в самом конце.

Тем не менее, водопадная модель часто интуитивно применяется при выполнении относительно простых задач, а ее недостатки послужили прекрасным отправным пунктом для создания новых моделей. Также эта модель в несколько усовершенствованном виде используется на крупных проектах, в которых требования очень стабильны и могут быть

хорошо сформулированы в начале проекта, например, аэрокосмическая область, медицинское программное обеспечение и т.д..

Итерационная инкрементальная модель

Итерационная инкрементальная модель (iterative model, incremental model) является фундаментальной основой современного подхода к разработке программного обеспечения. Как следует из названия модели, ей свойственна определенная двойственность. Глоссарий некоммерческой организации ISTQB, занимающейся определением различных принципов развития сферы тестирования программного обеспечения, даже не приводит единого определения, и разбивает его на отдельные части:

- с точки зрения жизненного цикла модель является итерационной, т.к. подразумевает многократное повторение одних и тех же стадий;
- с точки зрения развития продукта, т.е. приращения его полезных функций, модель является инкрементальной.

Ключевой особенностью данной модели является разбиение проекта на относительно небольшие промежутки, итерации, каждый из которых в общем случае может включать в себя все классические стадии, присущие водопадной модели. Итогом итерации является приращение, инкремент функциональности продукта, выраженное в промежуточной сборке, билде (build).

Итерационная инкрементальная модель очень хорошо зарекомендовала себя на объемных и сложных проектах, выполняемых большими командами на протяжении длительных сроков. Однако к основным недостаткам этой модели часто относят высокие накладные расходы, вызванные высокой «бюрократизированностью» и общей громоздкостью модели.

Гибкая модель

Гибкая модель (agile model) представляет собой совокупность различных подходов к разработке программного обеспечения и базируется на «agile-манифесте»:

1. Люди и взаимодействие важнее процессов и инструментов.
2. Работающий продукт важнее исчерпывающей документации.
3. Сотрудничество с заказчиком важнее согласования условий контракта.
4. Готовность к изменениям важнее следования первоначальному плану.

Как несложно догадаться, положенные в основу гибкой модели подходы являются логическим развитием и продолжением всего того, что было за десятилетия создано и опробовано в других моделях. Причем здесь впервые был достигнут ощутимый результат в снижении бюрократической составляющей и максимальной адаптации процесса разработки программного обеспечения к мгновенным изменениям рынка и требований заказчика.

Главным недостатком гибкой модели считается сложность ее применения к крупным проектам, а также частое ошибочное внедрение ее подходов, вызванное недопониманием фундаментальных принципов модели.

1.3. Качество программного обеспечения

Качество – это весьма субъективное и изменчивое понятие, что подтверждает следующее определение: «Качество – это ценность для отдельного значимого человека в определенный момент времени».

Первая часть определения принадлежит Вайнбергу Дж. (Jerry Weinberg). Она напоминает, что люди – сложные личности с уникальным опытом, и это приводит к разным представлениям о качестве. Пользователь А может считать, что ваш продукт высокого качества, в то время как пользователь Б с этим не согласится. При этом пользователь Б может быть для нас неважен и неинтересен, поэтому Бах Дж. (James Bach) и Болтон М. (Michael Bolton) добавили в определение часть «… **значимого** человека». Продукты, которые мы создаем, удовлетворяют конкретные потребности конкретных людей, поэтому определение качества должно зависеть от их взглядов на качество. Это значит, что предпочтение отдается людям, которые помогают нам держаться на плаву. Наконец, мы должны знать, что представление человека о качестве зависит от контекста, который меняется со временем, поэтому Гринлис Д. (David Greenlees) ввел в определение ссылку на время «…человека **в определенный момент времени**».

Понимание того, что качество является изменчивым и субъективным понятием, можно использовать при определении цели стратегии тестирования. Разобравшись в понимании качества нашими пользователями, мы можем определить, что надо тестировать, а что нет.

У каждого человека свое представление о качестве, на которое влияют прошлый опыт, предубеждения и повседневная жизнь. Как собрать и проанализировать информацию, чтобы понять это представление и преобразовать его в набор четких рабочих целей? Мы должны найти баланс: собрать достаточно подробную информацию о качестве, чтобы идти по правильному пути, но при этом не увязнуть в деталях, чтобы не перестать видеть лес за деревьями. Этого можно достичь путем систематизации характеристик качества.

Характеристика качества – это способ высокоуровневого описания какого-либо одного аспекта качества, например:

- Внешний вид и ощущения. Продукт считается качественным, если он хорошо выглядит или приятен в использовании. Он может иметь хорошо продуманную компоновку или изящный дизайн, облегчающий работу с ним. Например, для продукции Apple «внешний вид и ощущения» могут быть приоритетной характеристикой качества.
- Безопасность. Продукт считается качественным, если он обеспечивает защиту и безопасность информации. Это может быть надежное хранение данных, определенный уровень конфиденциальности или защита от нежелательного внимания. Например, менеджер паролей должен давать пользователям уверенность в безопасности.
- Точность. Продукт считается качественным, если он точно обрабатывает информацию. Он должен уметь работать со сложными процессами с множеством деталей и возвращать достоверные данные. Например, врач-терапевт рассматривает точность диагностики медицинским прибором как важную характеристику качества.

Характеристики качества многочисленны, и мы должны объединять их, чтобы получать надежную картину представления наших пользователей о качестве. Например, список характеристик, составленный Эдгреном Р. (Rikard Edgren), Эмильссоном Х. (Henrik Emilsson) и Янсоном М. (Martin Jansson) в работе «The Test Eye Software Quality Characteristics list», содержит более ста различных характеристик [http://thetesteye.com/posters/TheTestEye_SoftwareQualityCharacteristics.pdf].

Анализируя отзывы пользователей, можно определить, какие характеристики качества для них важнее, а затем использовать эти характеристики, чтобы определить цели стратегии тестирования. Это поможет увязать работу с задачей создания продукта, который считается качественным.

Если программный продукт создается по заказу конкретного клиента, тогда клиент может принимать в его проектировании самое непосредственное участие. Он предоставляет подробную спецификацию с описанием своих требований и собственного видения продукта, а разработчики соглашаются все это реализовать. В таком случае качество будет означать точное соответствие спецификации клиента.

У большинства разработчиков программного обеспечения таких грамотных и обстоятельных клиентов нет. Для них критерием качества служит не соответствие спецификации, а то, насколько пользователи удовлетворены программным продуктом и сопутствующими услугами компании. Если конечный результат пользователю не нравится, несмотря на наличие спецификации, значит, его качество не на высоте. И не важно, что пользователь ознакомился со спецификацией и согласился с ней или даже сам ее составил. Если, в конечном счете, продукт его не удовлетворяет, то только это и будет иметь для него значение.

Еще одной составляющей качества является надежность программного продукта. Надежность его тем выше, чем реже в нем происходят сбои, особенно такие, которые влекут за собой потерю данных и другие неприятные последствия. Хотя надежность

программы исключительно важна, она не является единственным критерием ее качества, и не правы те тестировщики, которые так думают. Если программа не позволяет пользователю выполнить что-то, что он считает важным, пользователь не будет ею доволен. А если пользователь недоволен, значит, качество программы нельзя назвать высоким.
Итак, качество программы определяется:
- возможностями, благодаря которым она понравится пользователю;
- недостатками, которые вынуждают пользователя приобрести другую программу.
Главное, что тестировщик может сделать для улучшения качества программы, – это выявить ее недостатки, сбои в ее работе и явные ошибки. Если руководитель проекта примет решение в последний момент добавить какую-нибудь очень важную функцию, это тоже может способствовать повышению качества, даже, несмотря на то, что от этого программа станет менее надежной. Ни надежность, ни функциональность программы не могут быть абсолютными, и ее качество, в конечном счете, означает разумный баланс между этими двумя характеристиками.

1.4. Связь тестирования с другими видами деятельности при разработке программного обеспечения

Тестировщики часто жалуются, что их подключают к процессу разработки слишком поздно, когда все принципиальные решения уже приняты. На самом же деле работа для тестировщика есть с самого начала – он может вносить полезные предложения на каждом этапе разработки. Например, на этапе разработки спецификации тестировщик может проводить анализ логической правильности спецификации, ее выполнимости, полезности отдельных решений и тестируемости программного продукта.
Серьезные программные продукты редко разрабатываются одиночками: обычно этим занимаются группы людей, иногда довольно многочисленные. В такой группе, называемой командой разработчиков (development team), у каждого сотрудника своя роль. Даже если приходится создавать программы абсолютно самостоятельно или в паре с коллегой, это просто означает, что участниками по очереди или одновременно выполняются функции всех необходимых членов команды.
Процесс разработки программного продукта можно разделить на несколько стадий. Весь этот нелегкий жизненный путь продукта, начиная от появления у авторов самой первой идеи и заканчивая его уходом со сцены, называется жизненным циклом (life cycle). В жизненном цикле продукта множество этапов.
Авторы Мартин Дж. и Мак-Клер К. (James Martin, Carma L. McClure) в своей книге Software Maintenance (Сопровождение программного обеспечения, 1983) привели относительную стоимость каждой их этих стадий (значения представлены в процентном отношении от общего бюджета проекта):
Стадия разработки
- Анализ требований 3%.
- Спецификация 3%.
- Проектирование 5%.
- Кодирование 7%.
- Тестирование 15%.
Производственная стадия
- Промышленное производство и сопровождение 67%.
Впервые эти цифры были опубликованы Зелковицем М., Шоу А. и Генноном Дж. (Marvin V. Zelkowitz, Alan C. Shaw, John D. Gannon) в 1979 году. По результатам их исследований и данным Мартина Дж. и Мак-Клера К., сопровождение программного продукта после его выпуска обходится дороже всего. На втором месте по стоимости стоит тестирование – на него приходится 45% всей стоимости разработки. В процессе сопровождения продукта при исправлении ошибок и внесении усовершенствований значительная часть затрат тоже приходится на тестирование.
Чем раньше найти и исправить ошибку, тем дешевле это обойдется!
На этапе планирования пока еще тестируются не программы – «тестируются» идеи. К их анализу привлекаются специалисты по маркетингу, руководители проекта, главные конструкторы и специалисты по анализу человеческого фактора. А вот члены группы тестирования участвуют в этой работе очень редко.
Если тестировщикам нужно будет выполнить анализ требований к продукту, следует опираться при анализе и оценке требований к продукту и его функциональных характеристик, прежде всего, пытайтесь выяснить следующее:
- Адекватны ли эти требования? Действительно ли именно такой продукт компания хочет создать?
- Полны ли они? Не упущены ли какие-нибудь еще полезные или даже жизненно необходимые функции? Нельзя ли ослабить какие-либо из перечисленных требований?
- Совместимы ли требования между собой? Требования к продукту и его функции могут оказаться логически или психологически несовместимыми. Логическая несовместимость означает их противоречивость, а психологическая – концептуальные разногласия, например, разобравшись с одной из функций, пользователь может не понять другую.
- Выполнимы ли они? Не требуется ли для нормальной эксплуатации продукта более быстрое аппаратное обеспечение, больший объем памяти, более высокая пропускная способность, большее разрешение и т.д., чем указано в документации?
- Разумны ли они? К сожалению, качество продукта и его рентабельность стоят по разную сторону баррикад: с одной стороны – производительность продукта, его надежность и нетребовательность к ресурсам, а с другой – время и стоимость его разработки. Найдено ли самое оптимальное соответствие между всеми этими характеристиками? Не требуется ли от продукта абсолютная безупречность, молниеносная работа и готовность конкурировать с программным обеспечением, которого еще нет и в проекте? Например, отдельные требования вполне достижимы, но не одновременно и не для одного и того же продукта. Поэтому одним из ключевых вопросов планирования является правильная расстановка приоритетов.
- Поддаются ли они тестированию? Насколько легко можно будет определить, соответствует ли инженерно-проектная документация требованиям к программному продукту.

1.5. Список основных терминов

Автоматизированное тестирование (automated testing) – набор техник, подходов и инструментальных средств, позволяющий исключить человека из выполнения некоторых задач в процессе тестирования.
Альфа-тестирование (alpha testing) – тестирование, которое выполняется внутри организации-разработчика с возможным частичным привлечением конечных пользователей. Может являться формой внутреннего приемочного тестирования.
Бета-тестирование (beta testing) – тестирование, которое выполняется вне организации-разработчика с активным привлечением конечных пользователей/заказчиков.
Граничное условие (border condition, boundary condition) – значение, находящееся на границе классов эквивалентности.
Дефект (defect, anomaly) – отклонение фактического результата от ожиданий наблюдателя, сформированных на основе требований, спецификаций, иной документации или опыта и здравого смысла.
Динамическое тестирование (dynamic testing) – тестирование с запуском кода на исполнение.
Дымовое тестирование (smoke test) – тестирование, которое направлено на проверку самой главной, самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной саму идею использования приложения (или иного объекта, подвергаемого дымовому тестированию).

Инспекция, аудит кода (code review, code inspection) – семейство техник повышения качества кода за счет того, что в процессе создания или совершенствования кода участвуют несколько человек. В отличие от техник статического анализа кода, аудит кода также улучшает такие его характеристики как понятность, поддерживаемость, соответствие соглашениям об оформлении и т.д. Аудит кода выполняется в основном самими программистами.
Интеграционное тестирование (integration testing) – тестирование, которое направлено на проверку взаимодействия между несколькими частями приложения (каждая из которых, в свою очередь, проверена отдельно на стадии модульного тестирования).
Класс эквивалентности (equivalence class) – набор данных, обрабатываемых одинаковым образом и приводящих к одинаковому результату.
Метод белого ящика (white box testing) – метод тестирования, в рамках которого у тестировщика есть доступ к внутренней структуре и коду приложения, а также есть достаточно знаний для понимания увиденного.
Метод серого ящика (gray box testing) – комбинация методов белого ящика и черного ящика, состоящая в том, что к части кода и архитектуры у тестировщика доступ есть, а к части – нет.
Метод черного ящика (black box testing) – метод тестирования, в рамках которого у тестировщика либо нет доступа к внутренней структуре и коду приложения, либо недостаточно знаний для их понимания, либо он сознательно не обращается к этим данным в процессе тестирования.
Метрика (metric) – числовая характеристика показателя качества. Может включать описание способов оценки и анализа результата.
Модель разработки программного обеспечения (software development model) – структура, систематизирующая различные виды проектной деятельности, их взаимодействие и последовательность в процессе разработки программного обеспечения. Выбор той или иной модели зависит от масштаба и сложности проекта, предметной области, доступных ресурсов и множества других факторов.
Модульное (компонентное) тестирование (unit testing, component testing) – тестирование, направленное на проверку отдельных небольших частей приложения, которые (как правило) можно исследовать изолированно от других подобных частей.
Набор тест-кейсов (test case suite, test suite, test set) – совокупность тест-кейсов, выбранных с некоторой общей целью или по некоторому общему признаку.
Негативное тестирование (negative testing) – тестирование, направленное на исследование работы приложения в ситуациях, когда с ним выполняются (некорректные) операции и/или используются данные, потенциально приводящие к ошибкам.
Нефункциональное тестирование (non-functional testing) – тестирование, направленное на проверку нефункциональных особенностей приложения (корректность реализации нефункциональных требований), таких как удобство использования, совместимость, производительность, безопасность и т.д.
Нефункциональные требования (non-functional requirements) – требования, описывающие свойства системы, удобство использования, безопасность, надежность, расширяемость и т.д., которыми система должна обладать при реализации своего поведения.
Отчет о дефекте (defect report) – документ, описывающий и приоритизирующий обнаруженный дефект, а также содействующий его устранению.
Отчет о результатах тестирования (test progress report, test summary report) - документ, обобщающий результаты работ по тестированию и содержащий информацию, достаточную для соотнесения текущей ситуации с тест-планом и принятия необходимых управленческих решений.
Отчетность (reporting) – сбор и распространение информации о результатах работы, включая текущий статус, оценку прогресса и прогноз развития ситуации.
Планирование (planning) – непрерывный процесс принятия управленческих решений и методической организации усилий по их реализации с целью обеспечения качества некоторого процесса на протяжении длительного периода времени.
Позитивное тестирование (positive testing) – тестирование, направленное на исследование приложения в ситуации, когда все действия выполняются строго по инструкции без ошибок, отклонений, ввода неверных данных и т.д.
Покрытие (coverage) – процентное выражение степени, в которой исследуемый элемент затронут соответствующим набором тест-кейсов.
Приемочное тестирование (acceptance testing) – формализованное тестирование, направленное на проверку приложения с точки зрения конечного пользователя/заказчика и вынесения решения о том, принимает ли заказчик работу у исполнителя, проектной команды.
Регрессионное тестирование (regression testing) – тестирование, направленное на проверку того факта, что в ранее работоспособной функциональности не появились ошибки, вызванные изменениями в приложении или среде его функционирования.
Ручное тестирование (manual testing) – тестирование, в котором тест-кейсы выполняются человеком вручную без использования средств автоматизации.
Системное тестирование (system testing) – тестирование, направленное на проверку всего приложения как единого целого, собранного из частей, проверенных на стадиях модульного и интеграционного тестирования.
Статическое тестирование (static testing) – тестирование без запуска кода на исполнение.
Тест (test) – набор из одного или нескольких тест-кейсов.
Тестирование критического пути (critical path test) – тестирование, направленное на исследование функциональности, используемой типичными пользователями в типичной повседневной деятельности.
Тестирование под управлением данными (data-driven testing) – способ разработки автоматизированных тест-кейсов, в котором входные данные и ожидаемые результаты выносятся за пределы тест-кейса и хранятся вне его – в файле, базе данных и т.д.
Тестирование под управлением ключевыми словами (keyword-driven testing) – способ разработки автоматизированных тест-кейсов, в котором за пределы тест-кейса выносится не только набор входных данных и ожидаемых результатов, но и логика поведения тест-кейса, которая описывается ключевыми словами.
Тестирование под управлением поведением (behavior-driven testing) – способ разработки автоматизированных тест-кейсов, в котором основное внимание уделяется корректности работы бизнес-сценариев, а не отдельным деталям функционирования приложения.
Тестирование программного обеспечения (software testing) – процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта.
Тестирование производительности (performance testing) – исследование показателей скорости реакции приложения на внешние воздействия при различной по характеру и интенсивности нагрузке.
Тест-кейс (test case) – набор входных данных, условий выполнения и ожидаемых результатов, разработанный с целью проверки того или иного свойства или поведения программного средства. Под тест-кейсом также может пониматься соответствующий документ, представляющий формальную запись тест-кейса.
Тест-план (test plan) – документ, описывающий и регламентирующий перечень работ по тестированию, а также соответствующие техники и подходы, стратегию, области ответственности, ресурсы, расписание и ключевые даты.
Требование (requirement) – описание того, какие функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи.
Трудозатраты (man-hours) – количество рабочего времени, необходимого для выполнения работы (выражается в человеко-часах).

Функциональная декомпозиция (functional decomposition) – процесс определения функции через ее разделение на низкоуровневые подфункции.
Функциональное тестирование (functional testing) – тестирование, направленное на проверку корректности работы функциональности приложения (корректность реализации функциональных требований).
Функциональные требования (functional requirements) – требования, описывающие поведение системы, т.е. ее действия, например, вычисления, преобразования, проверки, обработку и т.д..
Чек-лист (checklist) – набор идей. В общем случае чек-лист – это просто набор идей: идей по тестированию, идей по разработке, идей по планированию и управлению – любых идей.

2. ПРОЦЕССЫ ТЕСТИРОВАНИЯ И РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В литературе отмечается, что единого общепринятого набора классификаций не существует. Существуют схожие названия, которые предполагают такую классификацию по следующим уровням:
- по уровню тестирования, что предполагает рассмотрение уровней детализации приложения;
- по уровню функционального тестирования, основано на (убывании) степени важности тестируемых функций.

2.1. Особенности организации проведения тестирования с точки зрения используемых моделей разработки программного обеспечения

По мнению Канера С. работа руководителя проекта заключается в том, чтобы обеспечить выпуск высококачественного продукта в заданные сроки и при этом уложиться в рамки отпущенного бюджета. Чтобы выполнить свою задачу и удержать проект в рамках отпущенного времени и ресурсов, руководителю проекта в определенные моменты приходится пересматривать имеющиеся результаты работы и решать, какие из оставшихся задач должны быть обязательно выполнены, а от каких придется отказаться за неимением времени или денег.
В конечном счете, приходится поступаться следующим:
- надежностью продукта;
- количеством и разнообразием его функций;
- деньгами, необходимыми для выполнения дальнейшей работы;
- сроками выпуска продукта.

В принятии решения, особенно на поздних стадиях разработки, огромную роль играет гибкость выбранной стратегии разработки программного обеспечения.

Процесс тестирования при водопадной модели разработки

Следование водопадной модели разработки программного обеспечения на проекте для процесса тестирования требует уделять внимание следующим действиям.
- Как можно раньше проанализировать пользовательский интерфейс. Можно воспользоваться для этого прототипами программы.
- Как можно раньше начинать разработку тестового плана. Это позволит критически проанализировать спецификацию или ее черновики, и выявить возможные проблемы, связанные с тестированием будущего продукта.
- Нельзя приступить к тестированию раньше, чем продукт будет к нему готов. Проблема в том, что, если проектирование и кодирование продукта затянется, тестирование тоже будет начато позже срока. Поэтому следует тщательно продумывать действия в такой ситуации. Во-первых, необходимо чтобы сформированная группа квалифицированных тестировщиков не простаивала, и тестировщики должны приступить к работе, как только продукт будет готов. С другой стороны, если программа не будет достаточно стабильна, ее придется вернуть программистам на доработку, а тестировщиков загрузить другой работой, которая позволит ускорить процесс в дальнейшем. Например, они могут заняться автоматизацией тестов или формированием дополнительных файлов тестовых данных. Необходимо также разработать эффективную стратегию подключения к проекту дополнительных людей, если в последнюю минуту придется прибегнуть к такому варианту. Многие простые, но рутинные операции можно поручить неквалифицированному или не имеющему опыта работы с данной программой персоналу, повысив тем самым отдачу от основного состава группы. Однако задачи для подключаемых людей должны быть определены заранее, и вся схема действий должна быть достаточно продумана, поскольку, когда сроки начнут сильно поджимать, времени на это уже не будет.
- Тестирование – это наиболее ответственный этап разработки. К моменту начала тестирования программа уже практически полностью написана. После прекращения поиска ошибок она сразу же выходит в свет. Если проект достаточно велик для подключения нескольких тестировщиков, необходимо подумать о том, чтобы назначить одного-двух человек для постоянного выполнения неформального тестирования. Цель работы такого сотрудника – каждую неделю находить пару настолько серьезных ошибок или недостатков, что выпустить с ними продукт просто невозможно. Во время исправления этих ошибок остальная часть группы будет последовательно «прочесывать» программу в соответствии с планом. Эта тактика особенно хорошо зарекомендовала себя на больших и серьезных проектах, где она позволяет сэкономить очень много времени.

Процесс тестирования при эволюционной модели

При использовании на проекте эволюционной модели разработки программного обеспечения тестировщикам необходимо внимательно спланировать следующие действиям.
- Как можно раньше начинать тестирование. Как только будет сформирован базовый набор функций программы, необходимо будет приступить к тестированию ее надежности.
- Периодически проводить анализ функциональности и удобства программы. Если только что добавленная часть программы покажется неудачной, то будут все шансы добиться ее изменения.
- Вести разработку тестового плана параллельно с тестированием. Заранее разработать его не удастся – ведь не будет ни предмета тестирования, ни даже спецификации.
- Выполнять самую серьезную часть тестирования как можно раньше. Никогда не откладывать критические тесты «на потом». Руководитель проекта может в любое время остановить разработку, и они так и останутся невыполненными.

2.2. Жизненный цикл тестирования

Следуя общей логике итеративности, превалирующей во всех современных моделях разработки программного обеспечения, жизненный цикл тестирования также выражается замкнутой последовательностью действий:
1. Общее планирование и анализ требований.
2. Уточнение критериев приемки.
3. Уточнение стратегии тестирования.
4. Разработка тест-кейсов.
5. Выполнение тест-кейсов.
6. Фиксация найденных дефектов.
7. Анализ результатов тестирования.
8. Отчетность.

Общее планирование и анализ требований

Данная стадия объективно необходима как минимум для того, чтобы иметь ответ на такие вопросы, как:
- Что предстоит тестировать?
- Как много будет работы?
- Какие есть сложности?

- Все ли необходимое есть?

Как правило, получить ответы на эти вопросы невозможно без анализа требований, т.к. именно требования являются первичным источником ответов.

Уточнение критериев приемки

Данная стадия позволяет сформулировать или уточнить метрики и признаки возможности или необходимости начала тестирования (entry criteria), приостановки (suspension criteria) и возобновления (resumption criteria) тестирования, завершения или прекращения тестирования (exit criteria).

Уточнение стратегии тестирования

Стадия уточнение стратегии тестирования представляет собой еще одно обращение к планированию, но уже на локальном уровне: рассматриваются и уточняются те части стратегии тестирования (test strategy), которые актуальны для текущей итерации.

Разработка тест-кейсов

Данная стадия посвящена разработке, пересмотру, уточнению, доработке, переработке и прочим действиям с тест-кейсами, наборами тест-кейсов, тестовыми сценариями и иными артефактами, которые будут использоваться при непосредственном выполнении тестирования.

Выполнение тест-кейсов и фиксация найденных дефектов

Стадия выполнение тест-кейсов тесно связана со стадией фиксации найденных дефектов и фактически выполняются параллельно: дефекты фиксируются сразу по факту их обнаружения в процессе выполнения тест-кейсов. Однако зачастую после выполнения всех тест-кейсов и написания всех отчетов о найденных дефектах проводится явно выделенная стадия уточнения, на которой все отчеты о дефектах рассматриваются повторно с целью формирования единого понимания проблемы и уточнения таких характеристик дефекта, как важность и срочность.

Анализ результатов тестирования и отчетность

Стадия анализа результатов тестирования и стадия отчетности также тесно связаны между собой и выполняются практически параллельно. Формулируемые на стадии анализа результатов выводы напрямую зависят от плана тестирования, критериев приемки и уточненной стратегии. Полученные выводы оформляются и служат основой для стадий планирования тестирования, уточнения критериев приемки и стратегии следующей итерации тестирования.

2.3. Характеристики программного обеспечения

При определении качества программного обеспечения часто стараются свести это понятие к какому-то одному, показателю, например надежность функционирования программного обеспечения или программы. Это происходит вследствие традиционного подхода к решению качества программного обеспечения в рамках оценки как для обычной технической системы, представленной комплексом технических средств.

Процесс проектирования программного обеспечения начинается с определения требований к разрабатываемому программному обеспечению и его исходных данных. В результате анализа требований получают спецификацию программного обеспечения. В процессе определения спецификации конкретизируют основные функции программного обеспечения и его поведение.

Функциональные требования, характеристики

Функциональные требования описывают сервисы, предоставляемые программным обеспечением, его поведение в определенных ситуациях, реакцию на те или иные входные данные и действия, которые программное обеспечение позволит выполнять пользователям.

Функциональные требования документируются в спецификации требований к программному обеспечению, где описывается как можно более полно ожидаемое поведение системы.

Функциональная спецификация состоит из следующих частей.

- Описание внешней информационной среды, с которой будет взаимодействовать разрабатываемое программное обеспечение. Должны быть определены все используемые каналы ввода и вывода и все информационные объекты, к которым будет применяться разрабатываемое программное обеспечение, а также существенные связи между этими информационными объектами.
- Определение функций программного обеспечения, определенных на множестве состояний этой информационной среды. Вводятся обозначения всех определяемых функций, специфицируются их входные данные и результаты выполнения, с указанием типов данных и заданий всех ограничений, которых должны удовлетворять эти данные и результаты. Определяется содержание каждой из этих функций.
- Описание исключительных ситуаций, если таковые могут возникнуть при выполнении программ, и реакций на эти ситуации, которые должны обеспечить соответствующие программы. Должны быть перечислены все существующие случаи, когда программное обеспечение не может нормально выполнить, то или иную свою функцию. Для каждого такого случая должна быть определенная реакция программы.

Эксплуатационные требования, характеристики

Эксплуатационные требования определяют характеристики разрабатываемого программного обеспечения, проявляемые в процессе его использования.

К таким характеристикам относятся:

- Правильность – функционирование в соответствии с техническим заданием. Это требование является обязательным для всякого программного продукта, но поскольку никакое тестирование не дает полной гарантии правильности, то речь может идти об определенной вероятности наличия ошибок.
- Универсальность – обеспечение правильной работы при любых допустимых данных и защиты от неправильных данных. Поскольку доказать универсальность программы невозможно, то имеет смысл говорить о степени ее универсальности.
- Надежность – обеспечение полной повторяемости результатов, т.е. обеспечение их правильности при наличии различного рода сбоев. Источниками помех могут являться технические и программные средства, а также люди, работающие с этими средствами.
- Проверяемость – возможность проверки получаемых результатов. Для этого необходимо документально фиксировать исходные данные, установленные режимы и другую информацию, которая влияет на получаемые результаты.
- Точность результатов – обеспечение погрешности результатов не выше заданной.
- Защищенность – обеспечение конфиденциальности информации.
- Эффективность – использование минимально возможного количества ресурсов технических средств.

Четкие формулировки спецификации требований к разрабатываемому программному обеспечению – это задача достаточно сложная и ответственная, которая требует проведения предпроектных исследований.

2.4. Виды тестирования характеристик программного обеспечения

Тестирование можно классифицировать по очень большому количеству признаков. В нашем случае мы сосредоточимся на структуре организации программного обеспечения и связанного с ней поведении системы, которые можно назвать архитектурой программного обеспечения.

Архитектуру можно рекурсивно разобрать на части, взаимодействующие посредством интерфейсов, на связи, которые соединяют части, и на условия сборки частей.

Таким образом, тестирование можно классифицировать по следующим критериям:

- доступу к коду и архитектуре приложения;
- уровню детализации приложения;
- степени важности функций;
- принципам работы с приложением.

Доступ к коду и архитектуре приложения

Классификация тестирования программного обеспечения по доступу к коду и архитектуре приложения включает в себя:
- Метод белого (прозрачного, стеклянного) ящика.
- Метод черного ящика.
- Метод серого ящика.

Метод белого ящика

Метод белого ящика (white box testing, open box testing, clear box testing, glass box testing) – у тестировщика есть доступ к внутренней структуре и коду приложения, а также есть достаточно знаний для понимания увиденного. Тестирование кода возможно в ходе обычного просмотра, т.е. статическим тестированием, но так же и путем запуска код на выполнение. Например, модульное тестирование как раз и предполагает запуск кода на исполнение и при этом работу именно с кодом, а не с «приложением целиком».

Метод черного ящика

Метод черного ящика (black box testing) – у тестировщика либо нет доступа к внутренней структуре и коду приложения, либо недостаточно знаний для их понимания, либо он сознательно не обращается к ним в процессе тестирования. По такому принципу черного ящика работает большинство видов тестирования. Тестировщик оказывает на приложение воздействия и проверяет реакцию тем же способом, каким при реальной эксплуатации приложения на него воздействовали бы пользователи или другие приложения. В рамках тестирования по методу черного ящика основной информацией для создания тест-кейсов выступает документация, требования и общий здравый смысл, для случаев, когда поведение приложения в некоторой ситуации не регламентировано явно; иногда это называют «тестированием на основе неявных требований».

Метод серого ящика

Метод серого ящика (gray box testing) – комбинация методов белого ящика и черного ящика, состоящая в том, что к части кода и архитектуры у тестировщика доступ есть, а к части – нет.

Хочется отметить, что методы белого и черного ящика не являются конкурирующими или взаимоисключающими – напротив, они гармонично дополняют друг друга, компенсируя, таким образом, имеющиеся недостатки.

Уровень детализации программного обеспечения

Организация тестирования с точки зрения используемых модулей разработки программного обеспечения может быть представлена:
- модульным, компонентным тестированием;
- интеграционным тестированием;
- системным тестированием.

Модульное, компонентное тестирование

Модульное, компонентное тестирование (unit testing) направлено на проверку отдельных небольших частей приложения, которые можно исследовать изолированно от других подобных частей. При выполнении данного тестирования могут проверяться отдельные функции или методы классов, сами классы, взаимодействие классов, небольшие библиотеки, отдельные части приложения. Часто данный вид тестирования реализуется с использованием специальных технологий и инструментальных средств автоматизации тестирования, значительно упрощающих и ускоряющих разработку соответствующих тест-кейсов.

Интеграционное тестирование

Интеграционное тестирование (integration testing) направлено на проверку взаимодействия между несколькими частями приложения, каждая из которых, в свою очередь, проверена отдельно на стадии модульного тестирования. К сожалению, даже если работа ведется с очень качественными отдельными компонентами, «на стыке» их взаимодействия часто возникают проблемы. Именно эти проблемы и выявляет интеграционное тестирование.

Системное тестирование

Системное тестирование (system testing) направлено на проверку всего приложения как единого целого, собранного из частей, проверенных на двух предыдущих стадиях. Здесь не только выявляются дефекты «на стыках» компонентов, но и появляется возможность полноценно взаимодействовать с приложением с точки зрения конечного пользователя, применяя множество других видов тестирования.

Хочется отметить тот факт, что если предыдущая стадия обнаружила проблемы, то на следующей стадии эти проблемы точно скажутся на качестве, и если предыдущая стадия не обнаружила проблем, это еще не значит, что проблем не будет на следующей стадии.

Степень важности функций

По степени важности тестируемых функций или по уровню функционального тестирования, начиная с самого значимого и по убыванию важности, методы тестирования можно классифицировать следующим образом:
- дымовое тестирование;
- тестирование критического пути;
- расширенное тестирование.

Дымовое тестирование

Дымовое тестирование (smoke test) направлено на проверку самой главной, самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной саму идею использования приложения или иного объекта, подвергаемого дымовому тестированию.

Дымовое тестирование проводится после выхода нового билда, чтобы определить общий уровень качества приложения и принять решение о целесообразности или нет выполнения тестирования критического пути и расширенного тестирования. Поскольку тест-кейсов на уровне дымового тестирования относительно немного, а сами они достаточно просты, но при этом очень часто повторяются, они являются хорошими кандидатами на автоматизацию. В связи с высокой важностью тест-кейсов на данном уровне пороговое значение метрики их прохождения часто выставляется равным 100% или близким к 100%.

Тестирование критического пути

Тестирование критического пути (critical path test) направлено на исследование функциональности, используемой типичными пользователями в типичной повседневной деятельности. Именно эти функции и нужно проверить, как только мы убедились, что приложение «в принципе работает», когда дымовой тест прошел успешно. Если по каким-то причинам приложение не выполняет эти функции или выполняет их некорректно, очень многие пользователи не смогут достичь множества своих целей. Пороговое значение метрики успешного прохождения «теста критического пути» уже немного ниже, чем в дымовом тестировании, но все равно достаточно высоко, как правило, порядка 70-90% – в зависимости от сути проекта.

Расширенное тестирование

Расширенное тестирование (extended test) направлено на исследование всей заявленной в требованиях функциональности – даже той, которая низко проранжирована по степени важности. При этом здесь также учитывается, какая функциональность является более важной, а какая – менее важной. Но при наличии достаточного количества времени и иных ресурсов тест-кейсы этого уровня могут затронуть даже самые низкоприоритетные требования.

Еще одним направлением исследования в рамках данного тестирования являются нетипичные, маловероятные, экзотические случаи и сценарии использования функций и свойств приложения, затронутых на предыдущих уровнях. Пороговое значение метрики успешного прохождения расширенного тестирования существенно ниже, чем в тестировании критического пути и может составлять значения в диапазоне 30-50 %, т.к. подавляющее большинство найденных здесь дефектов не представляет угрозы для успешного использования приложения большинством пользователей.

3. ПЛАНИРОВАНИЕ И ОРГАНИЗАЦИЯ ПРОЦЕССА ТЕСТИРОВАНИЯ

3.1. Определение участников процесса тестирования

По мнению ChatGPT участники процесса тестирования могут включать различные роли, каждая из которых выполняет свои функции и задачи. Основные участники процесса тестирования в этом случае являются:

- Тест-менеджер – руководит процессом тестирования, разрабатывает стратегию тестирования, планирует ресурсы и сроки, управляет рисками и координирует работу команды тестировщиков.
- QA-инженеры (тестировщики) – выполняют тестирование программного обеспечения, разрабатывают и выполняют тестовые сценарии и тест-кейсы, фиксируют результаты тестирования и сообщают о найденных дефектах.
- Разработчики – исправляют выявленные дефекты, предоставляют техническую поддержку команде тестирования и могут участвовать в разработке автоматизированных тестов.
- Аналитики (бизнес-аналитики) – помогают в понимании требований к продукту, участвуют в разработке тестовых сценариев на основе бизнес-требований и обеспечивают связь между заказчиком и командой разработки.
- Менеджеры проектов – следят за соблюдением сроков и бюджета проекта, координируют взаимодействие между различными командами (разработки, тестирования, бизнеса) и обеспечивают выполнение проектных целей.
- Заказчики/пользователи – могут участвовать в приемочном тестировании, предоставляя обратную связь о функциональности продукта и его соответствии требованиям.
- Архитекторы – участвуют в проектировании системы и могут предоставлять рекомендации по улучшению архитектуры с точки зрения тестируемости.
- AQA-инженеры (специалисты по автоматизации тестирования) – разрабатывают автоматизированные тесты для повышения эффективности процесса тестирования и снижения времени на ручное тестирование.
- Специалисты по производительности (Performance Engineers) – проводят нагрузочное и стрессовое тестирование для оценки производительности системы под различными условиями нагрузки.
- Специалисты по безопасности (Security Testers) – оценивают безопасность приложения, выявляют уязвимости и проверяют соответствие стандартам безопасности.

Каждая из этих ролей играет важную роль в процессе обеспечения качества программного обеспечения, а их взаимодействие позволяет достичь высоких стандартов качества продукта.

В своей книге Блэк Р. отмечает, что некоторые ключевые процессы тестирования являются внутренними для группы тестирования и проводятся исключительно силами тестировщиков и влияют только на них. Другие процессы – совместные, в них включены другие отделы организации. Любой ключевой процесс зависит от контекста, в рамках которого он проходит и должен соответствовать участвующим в нем людям, системе, проекту и организации. Тестирование в ракурсе проекта связано с множеством различных видов деятельности и групп.

Например, рассмотрим процессы и группы, участвующие в процессе разработки программного обеспечения в ходе выполнения тестов. Выполнение тестов можно рассматривать как внутренний процесс, поскольку отслеживает результаты цикла тестирования. Предоставление отчетов об ошибках является совместным процессом, в ходе которого группа тестирования передает группе разработчиков информацию, получаемую в процессе проведения тестов. Отправка перечня изменений для конкретной тестовой версии представляет собой совместный процесс передачи группой разработчиков группе тестирования информации, требующейся в процессе проведения тестов. Поскольку процесс подготовки отчетов о состоянии тестирования зависит от контекста, тест-менеджер сообщает группе управления проектом о продвижении работ по тестированию и о том, как идет оценка качества программного продукта, используя графики и метрики, понятные данной группе и подходящие для нужд процесса разработки.

Многие передаваемые объекты и зависимости должны по времени и по форме сходиться в одной точке – на группе тестирования. Только в этом случае она сможет своевременно предоставить точную, полную и надлежащую информацию и соответствующие услуги по тестированию.

Кроме того, тесная связь между самими процессами тестирования и между процессами тестирования и другими процессами в проекте порождает взаимозависимость. Каждый процесс влияет на ход проведения и результат всех остальных процессов. В итоге устанавливаются петли обратной связи.

Вероятные участники процесса анализа рисков качества:

- Менеджер по маркетингу – оценивает ценность системы для заказчика или пользователя.
- Бизнес-аналитики, системные-аналитики – оценивают требования пользователей и основное предназначение системы.
- Менеджер по маркетингу, менеджер продукта, архитектор ПО – оценивают план развития продукта или системы.
- Архитектор ПО, менеджер разработки – оценивают техническую сторону долгосрочного развития.
- Бизнес-аналитик, системный аналитик, архитектор ПО, менеджер разработки, ведущий программист – оценивают ограничения системной архитектуры.
- Менеджер разработки, ведущий программист, программист, системный аналитик – оценивают ограничения реализации системы.
- Тест-менеджер, ведущий тестировщик – оценивают ограничения тестирования.
- Ведущий тестировщик, тестировщик – оценивают дефекты, обнаруженные в системах, разработанных ранее, сложные тестовые сценарии.
- Инженер сборок, менеджер сборок – оценивают проблемы, связанные с разработкой системы.
- Администраторы сети, системные администраторы – оценивают работу сети, аппаратного обеспечения.
- Менеджеры поддержки пользователей, менеджеры технической поддержки – оценивают параметры реальных пользователей, известные дефекты.
- Заказчик – оценивает проблемы, решение которых ожидается с помощью системы, или новые полезные возможности, которые привнесет система.
- Различные классы пользователей – оценивают, как пользователи принимают систему для решения задач или получения прибыли, каковы типичные наборы данных и сценарии использования системы, какие проблемы были у пользователей ранее.
- Специалист по качеству, менеджер группы разработки ПО, менеджер по качеству – оценивают гарантию согласованности процесса и качества в ходе разработки всей системы.
- Директор по качеству, генеральный директор, руководитель администрации, президент – оценивают насколько должен быть качественный продукт, чтобы его можно было выпустить под маркой компании.

По мнению Блэка Р. анализ рисков качества показывает, что наилучшие результаты достигаются, когда привлекается максимальное возможное число заинтересованных лиц с разными интересами и наличием квалификации в разных областях. Чем больше обнаруживается точек зрения в компании, тем сложнее перечень рисков и более точной будет расстановка приоритетов рисков.

Таким образом, список участников процесса тестирования может включать практически весь коллектив от разработчика, тестировщика, менеджера и до генерального директора. Однако реальная практика показывает, что на проектах за все отвечает команда тестирования, а иногда только один единственный тестировщик.

3.2. Основные документы при планировании процесса тестирования

В жизненном цикл тестирования каждая итерация начинается с планирования и заканчивается отчетностью, которая становится основой для планирования следующей итерации и так далее.

Таким образом, планирование и отчетность находятся в тесной взаимосвязи, и проблемы с одним из этих видов деятельности неизбежно приводят к проблемам с другим видом, а в конечном итоге и к проблемам с проектом в целом.

Планирование (planning) – непрерывный процесс принятия управленческих решений и методической организации усилий по их реализации с целью обеспечения качества некоторого процесса на протяжении длительного периода времени.

Отчетность (reporting) – сбор и распространение информации о результатах работы, включая текущий статус, оценку прогресса и прогноз развития ситуации.

Тест-план

Тест-план (test plan) – документ, описывающий и регламентирующий перечень работ по тестированию, а также соответствующие техники и подходы, стратегию, области ответственности, ресурсы, расписание и ключевые даты.

Ценность плана тестирования определяется тем, насколько он помогает в организации процесса тестирования и поиске ошибок.

В результате хороший тест-план обладает тремя важнейшими преимуществами:

- облегчает тестирование;
- помогает организовать взаимодействие между персоналом;
- представляет собой удобную структуру для организации, планирования и управления тестовым проектом.

Независимо от метода разработки продукта в целом, его тестирование и разработку тестового плана лучше всего выполнять эволюционным способом.

Эволюционный способ предполагает, что вместо того чтобы разрабатывать один большой тестовый план, работа начинается с одной из его составляющих и затем тест-план постепенно расширяется, путем добавления в него все новых и новых разделов.

Разработав первую часть тест-плана, можно приступить к поиску ошибок на его основе. Следующие разделы будут углублять и расширять области поиска ошибок. Лучше всего добавлять разделы в тест-план в порядке их приоритета, чтобы к тому моменту, когда руководитель объявит, что тестирование окончено и продукт выходит в свет, а это может случиться в любой момент, команда была уверена, что все наиболее важные тесты уже выполнены.

В общем случае тест-план включает следующие разделы.

- Цель. Предельно краткое описание цели разработки приложения, в частично это напоминает бизнес-требования, но здесь информация подается в еще более сжатом виде и в контексте того, на что следует обращать первостепенное внимание при организации тестирования и повышения качества.
- Области, подвергаемые тестированию. Перечень функций и функциональных особенностей приложения, которые будут подвергнуты тестированию. В некоторых случаях здесь также приводится приоритет соответствующей области.
- Области, не подвергаемые тестированию. Перечень функций и нефункциональных особенностей приложения, которые не будут подвергнуты тестированию. Причины исключения той или иной области из списка для тестирования могут быть самыми различными – от предельно низкой их важности для заказчика до нехватки времени или иных ресурсов. Этот перечень составляется, чтобы у проектной команды и иных заинтересованных лиц было четкое понимание, что тестирование таких-то особенностей приложения не запланировано – такой подход позволяет исключить появление ложных ожиданий и неприятных сюрпризов.
- Критерии (criteria). Этот раздел включает следующие подразделы:
  - Приемочные критерии, критерии качества (acceptance criteria) – любые объективные показатели качества, которым разрабатываемый продукт должен соответствовать с точки зрения заказчика или пользователя, чтобы считаться готовым к эксплуатации.
  - Критерии начала тестирования – перечень условий, при выполнении которых команда приступает к тестированию. Наличие этого критерия страхует команду от бессмысленной траты усилий в условиях, когда тестирование не принесет ожидаемой пользы.
  - Критерии приостановки тестирования – перечень условий, при выполнении которых тестирование приостанавливается. Наличие этого критерия также страхует команду от бессмысленной траты усилий в условиях, когда тестирование не принесет ожидаемой пользы.
  - Критерии возобновления тестирования – перечень условий, при выполнении которых тестирование возобновляется, как правило, после приостановки.
  - Критерии завершения тестирования – перечень условий, при выполнении которых тестирование завершается. Наличие этого критерия страхует команду, как от преждевременного прекращения тестирования, так и от продолжения тестирования в условиях, когда оно уже перестает приносить ощутимый эффект.
- Ресурсы. В данном разделе тест-плана перечисляются все необходимые для успешной реализации стратегии тестирования ресурсы: программные ресурсы, аппаратные ресурсы, человеческие ресурсы, временные ресурсы, финансовые ресурсы.
- Расписание (test schedule). Фактически это календарь, в котором указано, что и к какому моменту должно быть сделано. Особое внимание уделяется т.н. «ключевым точкам» (milestones), к моменту наступления которых должен быть получен некий значимый ощутимый результат.
- Роли и ответственность. Перечень необходимых ролей, например, «ведущий тестировщик», «эксперт по оптимизации производительности» и область ответственности специалистов, выполняющих эти роли.
- Оценка рисков. Перечень рисков, которые с высокой вероятностью могут возникнуть в процессе работы над проектом. По каждому риску дается оценка представляемой им угрозы, и приводятся варианты выхода из ситуации.
- Документация. Перечень используемой тестовой документации с указанием, кто и когда должен ее готовить и кому передавать.
- Метрики (metrics). Числовые характеристики показателей качества, способы их оценки, формулы и т.д. На этот раздел, как правило, формируется множество ссылок из других разделов тест-плана.

Метрики

Метрики в тестировании – это количественные показатели, используемые для оценки эффективности, качества и прогресса процесса тестирования. Они помогают принимать управленческие решения, выявлять узкие места и улучшать процессы.

Метрики могут быть как прямыми, которые не требуют вычислений, так и расчетными, которые вычисляются по формуле. Типичные примеры прямых метрик – количество разработанных тест-кейсов, количество найденных дефектов и т.д. В расчетных метриках могут использоваться как совершенно тривиальные, так и довольно сложные формулы.

В тестировании существует большое количество общепринятых метрик, многие из которых могут быть собраны автоматически с использованием инструментальных средств управления проектами. Например:

- процентное отношение (не) выполненных тест-кейсов ко всем имеющимся;
- процентный показатель успешного прохождения тест-кейсов;
- процентный показатель заблокированных тест-кейсов;
- плотность распределения дефектов;
- эффективность устранения дефектов;
- распределение дефектов по важности и срочности;

и т.д.
Тестовое покрытие – это одна из метрик оценки качества тестирования, представляющая плотность покрытия тестами требований либо исполняемого кода.
Покрытие (coverage) – процентное выражение степени, в которой исследуемый элемент (coverage item) затронут соответствующим набором тест-кейсов.
Сложность современного программного обеспечения и инфраструктуры сделало невыполнимой задачу проведения тестирования со 100% тестовым покрытием. Поэтому для разработки набора тестов, обеспечивающего более-менее высокий уровень покрытия можно использовать специальные инструменты либо техники тест дизайна.
Самыми простыми представителями метрик покрытия можно считать следующие метрики:
- Метрику покрытия требований. Требование считается «покрытым», если на него ссылается хотя бы один тест-кейс.
- Метрику плотности покрытия требований. В данной метрике учитывается, сколько тест-кейсов ссылается на несколько требований.
- Метрику покрытия классов эквивалентности, когда анализируется, сколько классов эквивалентности затронуто тест-кейсами.
- Метрику покрытия граничных условий, когда анализируется, сколько значений из группы граничных условий затронуто тест-кейсами.
Использование метрик помогает управлять качеством продукта и процессом тестирования, выявлять проблемные области и повышать эффективность работы команды.

3.3. Отчет о результатах тестирования

Отчет – это документ, содержащий информацию о выполненных действиях, результатах проведенной работы. Обычно он включает в себя таблицы, графики, списки, текст. В первую очередь необходимо понимать, для кого, для чего и в каких условиях мы это делаем и на сколько это улучшит восприятие излагаемой нами информации. Надо помнить, что каждое действие преследует определенную цель.
Ниже перечислены наиболее известные варианты отчетов в тестировании.
- Отчет по инциденту – документ, описывающий событие, которое произошло, например, во время тестирования, и которое необходимо исследовать. Отчет об инцидентах можно определить как письменное описание инцидента, наблюдаемого во время тестирования. Инцидент необходимо расследовать, и на основании расследования инцидент может быть преобразован в дефект. Чаще всего это оказывается дефектом, но иногда это может произойти из-за различных факторов, например: человеческий фактор, требование отсутствует или неясно, проблема среды, либо неправильная конфигурация среды, ошибочные тестовые данные, некорректный ожидаемый результат.
- Отчет о результатах тестирования – периодический отчет, в котором документируется подробная информация о выполнении теста и его результате. Также он содержит условия, предположения, ограничения теста. Помимо этого вносится подробная информация об оставшейся работе, чтобы показать, сколько еще работы необходимо выполнить в проекте.
- Отчет о выполнении теста содержит детали выполнения и результат выполнения теста. Обычно его готовят для отправки вышестоящему руководству от группы тестирования, чтобы показать состояние выполнения теста и ход тестирования. Когда мы доставляем программное обеспечение клиенту, мы вкратце отправим полную информацию о выполнении теста. Это даст клиенту лучшее понимание выполненной работы по тестированию.
- Отчет о ходе тестирования – документ, подводящий итог, составляемый с определенной периодичностью с целью сравнения прогресса тестирования с базовой версией, например, с исходным планом тестирования, и извещения о рисках и альтернативах, требующих решения руководства.
- Аналитический отчет о тестировании – документ, создаваемый в конце процесса тестирования и подводящий итог тестовым активностям и результатам. Также в нем содержится оценка процесса тестирования и полученный опыт.
- Отчет о результатах тестирования – документ, обобщающий результаты работ по тестированию и содержащий информацию, достаточную для соотнесения текущей ситуации с тест-планом и принятия необходимых управленческих решений.
В общем случае отчет о результатах тестирования включает следующие разделы.
- Краткое описание. В предельно краткой форме отражает основные достижения, проблемы, выводы и рекомендации. В идеальном случае прочтения краткого описания может быть достаточно для формирования полноценного представления о происходящем, что избавит от необходимости читать весь отчет.
- Команда тестировщиков. Список участников проектной команды, задействованных в обеспечении качества, с указанием их должностей и ролей в подотчетный период.
- Описание процесса тестирования. Последовательное описание работ, которые были выполнены за подотчетный период.
- Расписание. Детализированное расписание работы команды тестировщиков и/или личные расписания участников команды.
- Статистика по новым дефектам. Таблица, в которой представлены данные по обнаруженным за подотчетный период дефектам с классификацией по стадии жизненного цикла и важности.
- Список новых дефектов. Список обнаруженных за подотчетный период дефектов с их краткими описаниями и важностью.
- Статистика по всем дефектам. Таблица, в которой представлены данные по обнаруженным за все время существования проекта дефектам, с классификацией по стадии жизненного цикла и важности.
- Рекомендации. Обоснованные выводы и рекомендации по принятию тех или иных управленческих решений, например, по изменению тест-плана, запросу или освобождению ресурсов и т.д.
- Приложения. Фактические данные – это значения метрик и графическое представление их изменения во времени.

Лабораторная работа. Разработка и оформление плана тестирования.

Разработайте и оформите план тестирования веб-приложения на выбор:
- Интернет-магазин,
- сервис онлайн-продаж,
- сервис онлайн-заказов.
Опишите следующие проверки веб-приложения:
- функциональное тестирование,
- тестирование удобства пользования (usability, UI/UX),
- тестирование совместимости.
При описании функционального тестирования уделите внимание следующим опциям веб-приложения:
- проверка пользовательских форм,
- тестирование ссылок или проверка навигации сайта,
- тестирование форм регистраций и авторизаций,
- тестирование поисковой строки.
При описании тестирования удобства пользования веб-приложением обратите внимание на следующие моменты:
- проверка простоты эксплуатации,
- проверка удобства навигации,
- тестирование контента.
При тестировании на совместимость опишите, как веб-приложение должно смотреться на разных устройствах, будь то компьютер, ноутбук, планшет или мобильная версия.

Оформите план тестирования (test palan) согласно рекомендациям.

4. ТЕСТИРОВАНИЕ ТРЕБОВАНИЙ

4.1. Понятие о требованиях к программному обеспечению

Требование (requirement) – описание того, какие функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи.
Требования являются отправной точкой для определения того, что проектная команда будет проектировать, реализовывать и тестировать.
Поскольку мы постоянно говорим «документация и требования», а не просто «требования», то стоит рассмотреть перечень документации, которая должна подвергаться тестированию в процессе разработки программного обеспечения.
В общем случае документацию можно разделить на два больших вида в зависимости от времени и места ее использования:
- продуктная документация,
- проектная документация.

Продуктная документация (product documentation, development documentation) используется проектной командой во время разработки и поддержки продукта. Она включает:
- План проекта и в том числе тестовый план.
- Требования к программному продукту (product requirements) и функциональные спецификации (functional specifications).
- Архитектуру и дизайн программного обеспечения.
- Тест-кейсы и наборы тест-кейсов (test cases, test suites).
- Технические спецификации (technical specifications), такие как схемы баз данных, описания алгоритмов, интерфейсов и т.д.

Проектная документация (project documentation) включает в себя как продуктную документацию, так и некоторые дополнительные виды документации и используется не только на стадии разработки, но и на более ранних и поздних стадиях, например, на стадии внедрения и эксплуатации.
Проектная документация включает в себя:
- Пользовательскую и сопроводительную документацию, такую как встроенная помощь, руководство по установке и использованию, лицензионные соглашения и т.д.
- Маркетинговую документацию, которую представители разработчика или заказчика используют как на начальных этапах для уточнения сути и концепции проекта, так и на финальных этапах развития проекта для продвижения продукта на рынке.

В некоторых классификациях часть документов из продуктной документации может быть перечислена в проектной документации – это совершенно нормально, т.к. понятие проектной документации по определению является более широким.
Степень важности и глубина тестирования того или иного вида документации и отдельного документа определяется большим количеством факторов. Неизменным остается общий принцип – все, что мы создаем в процессе разработки проекта: рисунки маркером на доске, письма, переписка в мессенджерах, может считаться документацией и подвергать тестированию. Например, вычитывание письма перед отправкой – это тоже своего рода тестирование документации.

4.2. Выявление требований

Требования начинают свою жизнь на стороне заказчика. Их сбор и выявление осуществляются с помощью следующих основных техник:
- интервью,
- работа с фокусными группами,
- анкетирование,
- семинары и мозговой штурм,
- наблюдение,
- прототипирование,
- анализ документов,
- моделирование процессов и взаимодействий,
- самостоятельное описание.

Интервью – самый универсальный путь выявления требований, заключающийся в общении проектного специалиста, как правило, специалиста по бизнес-анализу и представителя заказчика или эксперта, пользователя и т.д. Интервью может проходить в виде беседы с вопросами и ответами, в виде переписки и т.п. Главным здесь является то, что ключевыми фигурами выступают двое – интервьюируемый и интервьюер, хотя это и не исключает наличия «аудитории слушателей», например, в виде лиц, поставленных в копию переписки.
Работа с фокусными группами может выступать как вариант «расширенного интервью», где источником информации является не одно лицо, а группа лиц, представляющих собой целевую аудиторию, обладающих важной для проекта информацией, уполномоченных принимать важные для проекта решения.
Анкетирование. Этот вариант выявления требований вызывает много споров, т.к. при неверной реализации может привести к нулевому результату при объемных затратах. В то же время при правильной организации анкетирование позволяет автоматически собрать и обработать огромное количество ответов от огромного количества респондентов.
Ключевым фактором успеха является правильное составление анкеты, правильный выбор аудитории и правильное преподнесение анкеты.
Семинары и мозговой штурм. Семинары позволяют группе людей очень быстро обменяться информацией и наглядно продемонстрировать те или иные идеи, а также хорошо сочетаются с интервью, анкетированием, прототипированием и моделированием – в том числе для обсуждения результатов и формирования выводов и решений. Мозговой штурм может проводиться и как часть семинара, и как отдельный вид деятельности. Он позволяет за минимальное время сгенерировать большое количество идей, которые в дальнейшем можно не спеша рассмотреть с точки зрения их использования для развития проекта.
Наблюдение. Может выражаться как в буквальном наблюдении за некими процессами, так и в вовлечении проектного специалиста в эти процессы в качестве участника. С одной стороны, наблюдение позволяет увидеть то, о чем могут умолчать интервьюируемые, анкетируемые и представители фокусных групп, но с другой – отнимает очень много времени и чаще всего позволяет увидеть лишь часть процессов.
Прототипирование состоит в демонстрации и обсуждении промежуточных версий продукта, например, дизайн страниц сайта может быть сначала представлен в виде картинок, и лишь затем сверстан. Это один из лучших путей поиска единого понимания и уточнения требований, однако он может привести к серьезным дополнительным затратам при отсутствии специальных инструментов, позволяющих быстро создавать прототипы, и слишком раннем применении, когда требования еще не стабильны, и высока вероятность создания прототипа, имеющего мало общего с тем, что хотел заказчик.
Анализ документов хорошо работает тогда, когда эксперты в предметной области временно недоступны, а также в предметных областях, имеющих регламентирующую анализируемую деятельность справочную документацию. Также к этой технике относится и просто изучение документов, регламентирующих бизнес-процессы в предметной области заказчика или в конкретной организации, что позволяет приобрести необходимые для лучшего понимания сути проекта знания.
Моделирование процессов и взаимодействий может применяться к бизнес-процессам и взаимодействиям, к техническим процессам и взаимодействиям. Данная техника требует высокой квалификации специалиста по бизнес-анализу, т.к. сопряжена с обработкой большого объема сложной и часто плохо структурированной информации.
Самостоятельное описание является не столько техникой выявления требований, сколько техникой их фиксации и формализации. Очень сложно пытаться самому «придумать

требования за заказчика», но в спокойной обстановке можно самостоятельно обработать собранную информацию и аккуратно оформить ее для дальнейшего обсуждения и уточнения.

4.3. Уровни, виды требований

Форма представления, степень детализации и перечень полезных свойств требований условно можно поделить на следующие уровни, виды:

- бизнес требования,
- пользовательские требования,
- бизнес правила,
- атрибуты качества,
- функциональные требования,
- нефункциональные требования,
- ограничения,
- требования к интерфейсам,
- требования к данным,
- спецификация требований.

Бизнес требования выражают цель, ради которой разрабатывается продукт, зачем он нужен, какая от него ожидается польза, как заказчик с его помощью будет получать прибыль. Результатом выявления требований на этом уровне является общее видение – документ, который, как правило, представлен простым текстом и таблицами. Здесь нет детализации поведения системы и иных технических характеристик, но вполне могут быть определены приоритеты решаемых бизнес-задач, риски и т.п.

Пользовательские требования описывают задачи, которые пользователь может выполнять с помощью разрабатываемой системы, реакцию системы на действия пользователя, сценарии работы пользователя. Поскольку здесь уже появляется описание поведения системы, требования этого уровня могут быть использованы для оценки объема работ, стоимости проекта, времени разработки и т.д. Пользовательские требования оформляются в виде вариантов использования, пользовательских историй, пользовательских сценариев.

Бизнес-правила описывают особенности принятых в предметной области и непосредственно у заказчика процессов, ограничений и иных правил. Эти правила могут относиться к бизнес-процессам, правилам работы сотрудников, нюансам работы программного обеспечения и т.д.

Атрибуты качества расширяют собой нефункциональные требования и на уровне пользовательских требований могут быть представлены в виде описания ключевых для проекта показателей качества, свойств продукта, не связанных с функциональностью, но являющихся важными для достижения целей создания продукта – производительность, масштабируемость, восстанавливаемость. Атрибутов качества очень много, но для любого проекта реально важными является лишь некоторое их подмножество.

Функциональные требования описывают поведение системы, т.е. ее действия. В контексте проектирования функциональные требования в основном влияют на дизайн системы. Стоит помнить, что к поведению системы относится не только то, что система должна делать, но и то, что она не должна делать.

Нефункциональные требования описывают свойства системы, которыми она должна обладать при реализации своего поведения, например, удобство использования, безопасность, надежность, расширяемость и т.д. В контексте проектирования нефункциональные требования в основном влияют на архитектуру системы.

Ограничения представляют собой факторы, ограничивающие выбор способов и средств, в том числе инструментов, реализации продукта.

Требования к интерфейсам описывают особенности взаимодействия разрабатываемой системы с другими системами и операционной средой.

Требования к данным описывают структуры данных и сами данные, являющиеся неотъемлемой частью разрабатываемой системы. Часто сюда относят описание базы данных и особенностей ее использования.

Спецификация требований объединяет в себе описание всех требований уровня продукта и может представлять собой весьма объемный документ сотни и тысячи страниц. Поскольку требований может быть очень много, а их приходится не только единожды написать и согласовать между собой, но и постоянно обновлять, работу проектной команды по управлению требованиями значительно облегчают соответствующие инструментальные средства.

4.4. Характеристики качественных требований

В процессе тестирования требований проверяется их соответствие определенному набору свойств:

- завершенность,
- атомарность,
- непротиворечивость,
- недвусмысленность,
- выполнимость,
- обязательность,
- прослеживаемость,
- модифицируемость,
- проранжированность,
- корректность.

Завершенность. Требование является полным и законченным с точки зрения представления в нем всей необходимой информации, ничто не пропущено по соображениям «это и так всем понятно».

Атомарность, единичность. Требование является атомарным, если его нельзя разбить на отдельные требования без потери завершенности и оно описывает одну и только одну ситуацию.

Непротиворечивость, последовательность. Требование не должно содержать внутренних противоречий и противоречий другим требованиям и документам.

Недвусмысленность. Требование должно быть описано без использования жаргона, неочевидных аббревиатур и расплывчатых формулировок, должно допускать только однозначное объективное понимание и быть атомарным в плане невозможности различной трактовки сочетания отдельных фраз.

Выполнимость. Требование должно быть технологически выполнимым и реализуемым в рамках бюджета и сроков разработки проекта.

Обязательность, нужность и актуальность. Если требование не является обязательным к реализации, оно должно быть просто исключено из набора требований. Если требование нужное, но «не очень важное», для указания этого факта используется указание приоритета. Также исключены или переработаны должны быть требования, утратившие актуальность.

Прослеживаемость. Прослеживаемость бывает вертикальной и горизонтальной. Вертикальная позволяет соотносить между собой требования на различных уровнях требований, горизонтальная позволяет соотносить требование с тест-планом, тест-кейсами, архитектурными решениями и т.д.

Для обеспечения прослеживаемости часто используются специальные инструменты по управлению требованиями или матрицы прослеживаемости (traceability matrix).

Модифицируемость. Это свойство характеризует простоту внесения изменений в отдельные требования и в набор требований. Можно говорить о наличии модифицируемости в том случае, если при доработке требований искомую информацию легко найти, а ее изменение не приводит к нарушению иных описанных в этом перечне свойств.

Проранжированность по важности, стабильности, срочности. Важность характеризует зависимость успеха проекта от успеха реализации требования. Стабильность характеризует вероятность того, что в обозримом будущем в требование не будет внесено никаких изменений. Срочность определяет распределение во времени усилий проектной команды по реализации того или иного требования.
Корректность и проверяемость. Фактически эти свойства вытекают из соблюдения всех вышеперечисленных. В дополнение можно отметить, что проверяемость подразумевает возможность создания объективного тест-кейса (тест-кейсов), однозначно показывающего, что требование реализовано верно и поведение приложения в точности соответствует требованию.

4.5. Техники тестирования требований

Тестирование документации и требований относится к разряду нефункционального тестирования. Основные техники такого тестирования в контексте требований таковы.
- Взаимный просмотр. Взаимный просмотр или рецензирование является одной из наиболее активно используемых техник тестирования требований и может быть представлен в одной из трех следующих форм, представленных по мере возрастания сложности и цены:
    - Беглый просмотр может выражаться как в показе автором своей работы коллегам с целью создания общего понимания и получения обратной связи, так и в простом обмене результатами работы между двумя и более авторами с тем, чтобы коллега высказал свои вопросы и замечания. Это самый быстрый, дешевый и часто используемый вид просмотра.
    - Технический просмотр (technical review) выполняется группой специалистов. В идеальной ситуации каждый специалист должен представлять свою область знаний. Тестируемый продукт не может считаться достаточно качественным, пока хотя бы у одного просматривающего остаются замечания.
    - Формальная инспекция представляет собой структурированный, систематизированный и документируемый подход к анализу документации. Для его выполнения привлекается большое количество специалистов, само выполнение занимает достаточно много времени, в результате этот вариант просмотра используется достаточно редко, как правило, при получении на сопровождение и доработку проекта, созданием которого ранее занималась другая компания.
- Вопросы. Следующей очевидной техникой тестирования и повышения качества требований после взаимного просмотра является отдельный вид деятельности, задавание вопросов. Если хоть что-то в требованиях вызывает у вас непонимание или подозрение – задавайте вопросы. Можно спросить представителей заказчика, можно обратиться к справочной информации. По многим вопросам можно обратиться к более опытным коллегам при условии, что у них имеется соответствующая информация, ранее полученная от заказчика. Главное, чтобы вопрос был сформулирован таким образом, чтобы полученный ответ позволил улучшить требования.
- Тест-кейсы и чек-листы. Хорошее требование является проверяемым, а значит, должны существовать объективные способы определения того, верно ли реализовано требование. Продумывание чек-листов или даже полноценных тест-кейсов в процессе анализа требований позволяет нам определить, насколько требование проверяемо. Если вы можете быстро придумать несколько пунктов чек-листа, это еще не признак того, что с требованием все хорошо. Например, требование может противоречить каким-то другим требованиям.

Но если никаких идей по тестированию требования в голову не приходит – это тревожный знак. Рекомендуется для начала убедиться, что вы понимаете требование. Для этого достаточно прочитать соседние требования, задать вопросы коллегам и т.д. Если нет, то можно пока отложить работу с данным конкретным требованием и вернуться к нему позднее – возможно, анализ других требований позволит вам лучше понять и это конкретное. Но если ничто не помогает – скорее всего, с требованием что-то не так.
Справедливости ради надо отметить, что на начальном этапе проработки требований такие случаи встречаются очень часто потому, что требования сформированы поверхностно, расплывчато и явно нуждаются в доработке, т.е. здесь нет необходимости проводить сложный анализ, чтобы констатировать непроверяемость требования.
- Рисунки (графическое представление). Чтобы увидеть общую картину требований целиком, очень удобно использовать рисунки, схемы, диаграммы, интеллект-карты и т.д. Графическое представление удобно одновременно своей наглядностью и краткостью, например, UML-схема базы данных, занимающая один экран, может быть описана несколькими десятками страниц текста. На рисунке очень легко заметить, что какие-то элементы «не стыкуются», что где-то чего-то не хватает и т.д. Если вы для графического представления требований будете использовать общепринятую нотацию, например, уже упомянутый UML, вы получите дополнительные преимущества: вашу схему смогут без труда понимать и дорабатывать коллеги, а в итоге может получиться хорошее дополнение к текстовой форме представления требований.
- Прототипирование. Можно сказать, что прототипирование часто является следствием создания графического представления и анализа поведения системы. С использованием специальных инструментов можно очень быстро сделать наброски пользовательских интерфейсов, оценить применимость тех или иных решений и даже создать не просто «прототип ради прототипа», а заготовку для дальнейшей разработки, если окажется, что реализованное в прототипе, возможно, с небольшими доработками устраивает заказчика.

4.6. Типичные ошибки при разработке и анализе требований

Рассмотрим типичные ошибки, совершаемые в процессе анализа и тестирования требований.
- Изменение формата файла и документа – если требования изначально создаются в некоей системе управления требованиями, этот вопрос неактуален, но высокоуровневые требования большинство заказчиков привыкли видеть в обычном docx-документе. Текстовой процессор Word предоставляет такие прекрасные возможности работы с документом, как отслеживание изменений и комментарии.
- Отметка того факта, что с требованием все в порядке. Если у вас не возникло вопросов или замечаний к требованию – не надо об этом писать. Любые пометки в документе подсознательно воспринимаются как признак проблемы, и такое «одобрение требований» только раздражает и затрудняет работу с документом – сложнее становится заметить пометки, относящиеся к проблемам.
- Описание одной и той же проблемы в нескольких местах. Помните, что ваши пометки, комментарии, замечания и вопросы тоже должны обладать свойствами хороших требований. Если вы много раз в разных местах пишете одно и то же об одном и том же, вы нарушаете как минимум свойство модифицируемости. Постарайтесь в таком случае вынести ваш текст в конец документа, укажите в его начале перечень пунктов требований, к которым он относится, а в самих требованиях в комментариях просто ссылайтесь на этот текст.
- Написание вопросов и комментариев без указания места требования, к которым они относятся. Если ваше инструментальное средство позволяет указать часть требования, к которому вы пишете вопрос или комментарий, сделайте это, например, текстовый процессор Word позволяет выделить для комментирования любую часть текста. Если это невозможно, цитируйте соответствующую часть текста. В противном случае вы порождаете неоднозначность или вовсе делаете вашу пометку бессмысленной, т.к. становится невозможно понять, о чем вообще идет речь.
- Задавание плохо сформулированных вопросов. Эта ошибка была подробно рассмотрена выше. Однако добавим, что есть еще три вида плохих вопросов:

- Первый вид возникает из-за того, что автор вопроса не знает общепринятой терминологии или типичного поведения стандартных элементов интерфейса, например, «что такое чек-бокс?», «как в списке можно выбрать несколько пунктов?», «как подсказка может всплывать?».
- Второй вид плохих вопросов похож на первый из-за формулировок: вместо того, чтобы написать «что вы имеете в виду под «чем-то?»», автор вопроса пишет «что такое», «что-то?» То есть вместо вполне логичного уточнения получается ситуация, очень похожая на рассмотренную в предыдущем пункте.
- Третий вид сложно привязать к причине возникновения, но его суть в том, что к некорректному или невыполнимому требованию задается вопрос «что будет, если мы это сделаем?». Ничего не будет, т.к. мы это точно не сделаем. И вопрос должен быть совершенно иным, каким именно – зависит от конкретной ситуации, но точно не таким.

И еще раз напомним о точности формулировок: иногда одно-два слова могут уничтожить отличную идею, превратив хороший вопрос в плохой. Сравните: «Что такое формат даты по умолчанию?» и «Каков формат даты по умолчанию?». Первый вариант просто показывает некомпетентность автора вопроса, тогда как второй – позволяет получить полезную информацию.

К этой же проблеме относится непонимание контекста. Часто можно увидеть вопросы в стиле «о каком приложении идет речь?», «что такое система?» и им подобные. Чаще всего автор таких вопросов просто вырвал требование из контекста, по которому было совершенно ясно, о чем идет речь.

- Написание очень длинных комментариев или вопросов. История знает случаи, когда одна страница исходных требований превращалась в 20–30 страниц текста анализа и вопросов. Это плохой подход. Все те же мысли можно выразить значительно более кратко, чем сэкономить как свое время, так и время автора исходного документа. Тем более стоит учитывать, что на начальных стадиях работы с требованиями они весьма нестабильны, и может получиться так, что ваши 5–10 страниц комментариев относятся к требованию, которое просто удалят или изменят до неузнаваемости.
- Критика текста или даже его автора. Помните, что ваша задача – сделать требования лучше, а не показать их недостатки или недостатки автора. Потому комментарии вида «плохое требование», «неужели вы не понимаете, как глупо это звучит», «надо переформулировать» неуместны и недопустимы.
- Категоричные заявления без обоснования. Как продолжение ошибки «критика текста или даже его автора» можно отметить и просто категоричные заявления «это невозможно», «мы не будем этого делать», «это не нужно». Даже если вы понимаете, что требование бессмысленно или невыполнимо, эту мысль стоит сформулировать в корректной форме и дополнить вопросами, позволяющими автору документа самому принять окончательное решение.

Например, «это не нужно» можно переформулировать так: «Мы сомневаемся в том, что данная функция будет востребована пользователями. Какова важность этого требования? Уверены ли вы в его необходимости?»

- Указание проблемы с требованиями без пояснения ее сути. Помните, что автор исходного документа может не быть специалистом по тестированию или бизнес-анализу. Потому просто пометка в стиле «неполнота», «двусмысленность» и т.д. могут ничего ему не сказать. Поясняйте свою мысль.

Сюда же можно отнести небольшую, но досадную недоработку, относящуюся к противоречивости: если вы обнаружили некие противоречия, сделайте соответствующие пометки во всех противоречащих друг другу местах, а не только в одном из них. Например, вы обнаружили, что требование 20 противоречит требованию 30. Тогда в требовании 20 отметьте, что оно противоречит требованию 30, и наоборот. И поясните суть противоречия.

- Плохое оформление вопросов и комментариев. Старайтесь сделать ваши вопросы и комментарии максимально простыми для восприятия. Помните не только о краткости формулировок, но и об оформлении текста. Например, когда вопросы структурированы в виде списка – такая структура воспринимается намного лучше, чем сплошной текст. Перечитайте свой текст, исправьте опечатки, грамматические и пунктуационные ошибки и т.д.
- Описание проблемы не в том месте, к которому она относится. Классическим примером может быть неточность в сноске, приложении или рисунке, которая почему-то описана не там, где она находится, а в тексте, ссылающемся на соответствующий элемент. Исключением может считаться противоречивость, при которой описать проблему нужно в обоих местах.
- Ошибочное восприятие требования как «требования к пользователю». Требования в стиле «пользователь должен быть в состоянии отправить сообщение» являются некорректными. И это так. Но бывают ситуации, когда проблема намного менее опасна и состоит только в формулировке. Например, фразы в стиле «пользователь может нажать на любую из кнопок», «пользователю должно быть видно главное меню» на самом деле означают «все отображаемые кнопки должны быть доступны для нажатия» и «главное меню должно отображаться». Да, эту недоработку тоже стоит исправить, но не следует отмечать ее как критическую проблему.
- Скрытое редактирование требований. Эту ошибку можно смело отнести к разряду крайне опасных. Ее суть состоит в том, что тестировщик произвольно вносит правки в требования, никак не отмечая этот факт. Соответственно, автор документа, скорее всего, не заметит такой правки, а потом будет очень удивлен, когда в продукте что-то будет реализовано совсем не так, как когда-то было описано в требованиях. Потому простая рекомендация: если вы что-то правите, обязательно отмечайте это средствами вашего инструмента или просто явно в тексте. И еще лучше отмечать правку как предложение по изменению, а не как свершившийся факт, т.к. автор исходного документа может иметь совершенно иной взгляд на ситуацию.
- Анализ, не соответствующий уровню требований. При тестировании требований следует постоянно помнить, к какому уровню они относятся, т.к. в противном случае появляются следующие типичные ошибки:
  - добавление в бизнес-требования мелких технических подробностей;
  - дублирование на уровне пользовательских требований части бизнес-требований, если вы хотите увеличить прослеживаемость набора требований, имеет смысл просто использовать ссылки;
  - недостаточная детализация требований уровня продукта, общие фразы, допустимые, например, на уровне бизнес-требований, здесь уже должны быть предельно детализированы, структурированы и дополнены подробной технической информацией.

Лабораторная работа. Тестирование требований к программному обеспечению.

Разработайте и оформите требования к отдельному модулю:
- форма регистрации,
- форма авторизаций,

веб-приложения на выбор:
- Интернет-магазин,
- сервис онлайн-продаж,
- сервис онлайн-заказов.

В ходе разработки требований используйте возможности текстового процессора Word для хранения версий документа, комментариев и рецензирования.

5. РАЗРАБОТКА ТЕСТОВ И НАБОРОВ ТЕСТОВ

5.1. Аксиомы тестирования

Описание предполагаемых значений выходных данных или результатов должно быть необходимой частью тестового набора. Тест должен включать две компоненты: описание входных данных и описание точного и корректного результата, соответствующего набору входных данных.
Следует избегать тестирования программ ее автором. Однако это не значит, что программист не может тестировать свою программу. Здесь лишь делается предположение, что тестирование является более эффективным, если оно выполняется кем-либо другим. Программирующая организация не должна сама тестировать разработанные ею программы.
Необходимо досконально изучать результаты применения каждого теста. Тесты для неправильных и непредусмотренных входных данных следует разрабатывать так же тщательно, как для правильных и предусмотренных. Необходимо проверять не только, делает ли программа то, для чего она предназначена, но и не делает ли она то, что не должна делать.
Не следует выбрасывать тесты, даже если программа уже не нужна. Нельзя планировать тестирование в предположении, что ошибки не будут обнаружены. Вероятность наличия необнаруженных ошибок в части программы пропорциональна числу ошибок, уже обнаруженных в этой части.
Тестирование – процесс творческий. В заключение хочется еще раз повторить некоторые мысли, которые являются наиболее важными принципами тестирования:
- Тестирование – это процесс выполнения программ с целью обнаружения ошибок.
- Хорошим считается тест, который имеет высокую вероятность обнаружения еще не выявленной ошибки.
- Удачным считается тест, который обнаруживает еще не выявленную ошибку.

5.2. Чек-листы. Тест-кейсы: понятие, структура

Чек-листы

Чек-лист (checklist) – набор идей (тест-кейсов). В общем случае чек-лист – это просто набор идей: идей по тестированию, идей по разработке, идей по планированию и управлению – любых идей.
Чек-лист чаще всего представляет собой обычный и привычный нам список:
- в котором последовательность пунктов не имеет значения, например, список значений некоего поля для ввода информации;
- в котором последовательность пунктов важна, например, шаги в краткой инструкции;
- структурированный, многоуровневый список, который позволяет отразить иерархию идей.

Для того чтобы чек-лист был действительно полезным инструментом, он должен обладать рядом важных свойств.
- Логичность. Чек-лист пишется не «просто так», а на основе целей и для того, чтобы помочь в достижении этих целей. К сожалению, одной из самых частых и опасных ошибок при составлении чек-листа является превращение его в свалку мыслей, которые никак не связаны друг с другом.
- Последовательность и структурированность. Со структурированностью все достаточно просто – она достигается за счет оформления чек-листа в виде многоуровневого списка. Человеку удобно воспринимать информацию в виде неких небольших групп идей, переход между которыми является понятным и очевидным. Например, сначала можно прописать идеи простых позитивных тест-кейсов, потом идеи простых негативных тест-кейсов, потом постепенно повышать сложность тест-кейсов, но не стоит писать эти идеи вперемешку.
- Полнота и неизбыточность. Чек-лист должен представлять собой аккуратную «сухую выжимку» идей, в которых нет дублирования, которое часто появляется из-за разных формулировок одной и той же идеи, и в то же время ничто важное не упущено.

Правильно созданные и оформленные чек-листы также помогают восприятию их не только как хранилища наборов идей, но и как «требования» к программному обеспечению. Эта мысль приводит к пересмотру и переосмыслению свойств качественных требований в применении к чек-листам.

Структура тест-кейса

Термин «тест-кейс» может относиться к формальной записи тест-кейса в виде технического документа. Эта запись имеет общепринятую структуру, компоненты которой называются атрибутами, полями тест-кейса.
В зависимости от инструмента управления тест-кейсами внешний вид их записи может немного отличаться, могут быть добавлены или убраны отдельные поля, но концепция остается неизменной.
Перечислим основные атрибуты, поля тест-кейса.
- Идентификатор (identifier) представляет собой уникальное значение, позволяющее однозначно отличить один тест-кейс от другого и используемое во всевозможных ссылках. В общем случае идентификатор тест-кейса может представлять собой просто уникальный номер, но, если позволяет инструментальное средство управления тест-кейсами, то может быть и куда сложнее: включать префиксы, суффиксы и иные осмысленные компоненты, позволяющие быстро определить цель тест-кейса и часть приложения или требований, к которой он относится. Например: UR216_S12_DB_Neg.
- Приоритет (priority) показывает важность тест-кейса. Он может быть выражен буквами (A, B, C, D, E), цифрами (1, 2, 3, 4, 5), словами («крайне высокий», «высокий», «средний», «низкий», «крайне низкий») или иным удобным способом. Количество градаций также не фиксировано, но чаще всего лежит в диапазоне от трех до пяти.

Приоритет тест-кейса может коррелировать с:
- важностью требования, пользовательского сценария или функции, с которыми связан тест-кейс;
- потенциальной важностью дефекта, на поиск которого направлен тест-кейс;
- степенью риска, связанного с проверяемым тест-кейсом требованием, сценарием или функцией.

Основная задача этого атрибута – упрощение распределения внимания и усилий команды (более высокоприоритетные тест-кейсы получают их больше), а также упрощение планирования и принятия решения о том, чем можно пожертвовать в некоей форс-мажорной ситуации, не позволяющей выполнить все запланированные тест-кейсы.
- Связанное с тест-кейсом требование (requirement) показывает то основное требование, проверке выполнения которого посвящен тест-кейс. Основное – потому, что один тест-кейс может затрагивать несколько требований. Наличие этого поля улучшает такое свойство тест-кейса, как прослеживаемость.
- Модуль и подмодуль приложения указывают на части приложения, к которым относится тест-кейс, и позволяют лучше понять его цель.
- Заглавие тест-кейса (title) призвано упростить и ускорить понимание основной идеи, цели тест-кейса без обращения к его остальным атрибутам. Именно это поле является наиболее информативным при просмотре списка тест-кейсов.
- Исходные данные, необходимые для выполнения тест-кейса (precondition, preparation, initial data, setup), позволяют описать все то, что должно быть подготовлено до начала выполнения тест-кейса.
- Шаги тест-кейса (steps) описывают последовательность действий, которые необходимо реализовать в процессе выполнения тест-кейса.
- Ожидаемые результаты (expected results) по каждому шагу тест-кейса описывают реакцию приложения на действия, описанные в поле «шаги тест-кейса». Номер шага соответствует номеру результата.

5.3. Характеристики качественных тест-кейсов

Правильно оформленный тест-кейс может оказаться некачественным, если в нем нарушено одно из следующих свойств.
- Правильный технический язык, точность и единообразие формулировок. Это свойство в равной мере относится и к требованиям, и к тест-кейсам, и к отчетам о дефектах – к любой документации.
- Баланс между специфичностью и общностью. Тест-кейс считается тем более специфичным, чем более детально в нем расписаны конкретные действия, конкретные значения и т.д., т.е. чем в нем больше конкретики. Соответственно, тест-кейс считается тем более общим, чем в нем меньше конкретики.
- Баланс между простотой и сложностью. Принято считать, что простой тест-кейс оперирует одним объектом или в нем явно виден главный объект, а также содержит небольшое количество тривиальных действий; сложный тест-кейс оперирует несколькими равноправными объектами и содержит много нетривиальных действий.
- «Показательность» (высокая вероятность обнаружения ошибки). Начиная с уровня тестирования критического пути, можно утверждать, что тест-кейс является тем более хорошим, чем он более показателен, т.е. с большей вероятностью обнаруживает ошибку. Именно поэтому можно считать непригодными слишком простые тест-кейсы – они непоказательны.
- Последовательность в достижении цели. Суть этого свойства выражается в том, что все действия в тест-кейсе направлены на следование единой логике и достижение единой цели и не содержат никаких отклонений.
- Неизбыточность по отношению к другим тест-кейсам. В процессе создания множества тест-кейсов очень легко оказаться в ситуации, когда два и более тест-кейса фактически выполняют одни и те же проверки, преследуют одни и те же цели, направлены на поиск одних и тех же проблем.
- Демонстративность – способность демонстрировать обнаруженную ошибку очевидным образом. Ожидаемые результаты должны быть подобраны и сформулированы таким образом, чтобы любое отклонение от них сразу же бросалось в глаза и становилось очевидным, что произошла ошибка.
- Прослеживаемость. Из содержащейся в качественном тест-кейсе информации должно быть понятно, какую часть приложения, какие функции и какие требования он проверяет. Частично это свойство достигается через заполнение соответствующих полей тест-кейса, но и сама логика тест-кейса играет не последнюю роль, т.к. в случае серьезных нарушений этого свойства можно долго с удивлением смотреть, например, на какое требование ссылается тест-кейс, и пытаться понять, как же они друг с другом связаны.
- Возможность повторного использования. Это свойство редко выполняется для низкоуровневых тест-кейсов, но при создании высокоуровневых тест-кейсов можно добиться таких формулировок, при которых:
    - тест-кейс будет пригодным к использованию с различными настройками тестируемого приложения и в различных тестовых окружениях;
    - тест-кейс практически без изменений можно будет использовать для тестирования аналогичной функциональности в других проектах или других областях приложения.
- Повторяемость. Тест-кейс должен быть сформулирован таким образом, чтобы при многократном повторении он показывал одинаковые результаты. Это свойство можно разделить на два подпункта:
    - даже общие формулировки, допускающие разные варианты выполнения тест-кейса, должны очерчивать соответствующие явные границы;
    - действия (шаги) тест-кейса по возможности не должны приводить к необратимым или сложно обратимым последствиям – не стоит включать в тест-кейс такие «разрушительные действия», если они не продиктованы явным образом целью тест-кейса; если же цель тест-кейса обязывает нас к выполнению таких действий, в самом тест-кейсе должно быть описание действий по восстановлению исходного состояния приложения (данных, окружения).
- Соответствие принятым шаблонам оформления и традициям. С шаблонами оформления, как правило, проблем не возникает: они строго определены имеющимся образцом или вообще экранной формой инструментального средства управления тест-кейсами.

5.4. Наборы тест-кейсов

Набор тест-кейсов (test case suite, test suite, test set) – совокупность тест-кейсов, выбранных с некоторой общей целью или по некоторому общему признаку. Иногда в такой совокупности результаты завершения одного тест-кейса становятся входным состоянием приложения для следующего тест-кейса.

В общем случае наборы тест-кейсов можно разделить на свободные, когда порядок выполнения тест-кейсов не важен, и последовательные, когда порядок выполнения тест-кейсов важен.

Преимущества свободных наборов:
- Тест-кейсы можно выполнять в любом удобном порядке, а также создавать «наборы внутри наборов».
- Если какой-то тест-кейс завершился ошибкой, это не повлияет на возможность выполнения других тест-кейсов.

Преимущества последовательных наборов:
- Каждый следующий в наборе тест-кейс в качестве входного состояния приложения получает результат работы предыдущего тест-кейса, что позволяет сильно сократить количество шагов в отдельных тест-кейсах.
- Длинные последовательности действий куда лучше имитируют работу реальных пользователей, чем отдельные «точечные» воздействия на приложение.

К отдельному подвиду последовательных наборов тест-кейсов или даже неоформленных идей тест-кейсов, таких, как пункты чек-листа можно отнести пользовательские сценарии или сценарии использования, представляющие собой цепочки действий, выполняемых пользователем в определенной ситуации для достижения определенной цели.

Классификация наборов тест-кейсов может быть представлена:
- Набор изолированных свободных тест-кейсов: действия из раздела «приготовления» нужно повторить перед каждым тест-кейсом, а сами тест-кейсы можно выполнять в любом порядке.
- Набор обобщенных свободных тест-кейсов: действия из раздела «приготовления» нужно выполнить один раз, а потом просто выполнять тест-кейсы, и сами тест-кейсы можно выполнять в любом порядке.
- Набор изолированных последовательных тест-кейсов: действия из раздела «приготовления» нужно повторить перед каждым тест-кейсом, а сами тест-кейсы нужно выполнять в строго определенном порядке.
- Набор обобщенных последовательных тест-кейсов: действия из раздела «приготовления» нужно выполнить один раз, а потом просто выполнять тест-кейсы, и сами тест-кейсы нужно выполнять в строго определенном порядке.

Главное преимущество изолированности: каждый тест-кейс выполняется в «чистой среде», на него не влияют результаты работы предыдущих тест-кейсов.

Главное преимущество обобщенности: приготовления не нужно повторять.

Главное преимущество последовательности: ощутимое сокращение шагов в каждом тест-кейсе, т.к. результат выполнения предыдущего тест-кейса является начальной ситуацией для следующего.

Главное преимущество свободы: возможность выполнять тест-кейсы в любом порядке, а также то, что при провале некоего тест-кейса когда приложение не пришло в ожидаемое состояние, остальные тест-кейсы по-прежнему можно выполнять.

Лабораторная работа. Разработка и оформление тестов и наборов тестов.

Разработайте и оформите тесты для функционального тестирования веб-приложения на выбор:
- форма регистрации,
- форма авторизации,
- корзина заказов, покупок,
- форма онлайн-заказа.
Оформите тесты согласно рекомендациям.

6. АНАЛИЗ РЕЗУЛЬТАТОВ ТЕСТИРОВАНИЯ

Отчет о дефекте (defect report) – документ, описывающий обнаруженный дефект, а также содействующий его устранению. Как следует из самого определения, отчет о дефекте пишется со следующими основными целями:
- предоставить информацию о проблеме – уведомить проектную команду и иных заинтересованных лиц о наличии проблемы, описать суть проблемы;
- указать приоритет для проблемы – определить степень опасности проблемы для проекта и желаемые сроки ее устранения;
- содействовать устранению проблемы – качественный отчет о дефекте не только предоставляет все необходимые подробности для понимания сути случившегося, но также может содержать анализ причин возникновения проблемы и рекомендации по исправлению ситуации.

6.1. Понятие дефект

Дефект – расхождение ожидаемого и фактического результата.
Ожидаемый результат – поведение системы, описанное в требованиях.
Фактический результат – поведение системы, наблюдаемое в процессе тестирования.
В силлабусе ISTQB написано, что человек совершает ошибки, которые приводят к возникновению дефектов в коде, которые, в свою очередь, приводят к сбоям и отказам приложения. Однако сбои и отказы могут возникать и из-за внешних условий, таких как электромагнитное воздействие на оборудование и т.д.
Ошибка (error, mistake) – действие человека, приводящее к некорректным результатам.
Дефект (defect, bug, problem, fault) – недостаток в компоненте или системе, способный привести к ситуации сбоя или отказа.
Сбой (interruption) или отказ (failure) – отклонение поведения системы от ожидаемого.
В ГОСТ 27.002-89 даны хорошие и краткие определения сбоя и отказа.
Сбой – самоустраняющийся отказ или однократный отказ, устраняемый незначительным вмешательством оператора.
Отказ – событие, заключающееся в нарушении работоспособного состояния объекта.
Аномалия (anomaly) или инцидент (incident, deviation) – любое отклонение наблюдаемого, фактического состояния, поведения, значения, результата, свойства от ожиданий наблюдателя, сформированных на основе требований, спецификаций, иной документации или опыта и здравого смысла.
Таким образом, дефект – отклонение фактического результата (actual result) от ожиданий наблюдателя (expected result), сформированных на основе требований, спецификаций, иной документации или опыта и здравого смысла.

6.2. Классификация дефектов, обнаруженных при тестировании

Основные виды дефектов можно классифицировать по различным признакам.

По степени важности и влиянию
- Критические (Critical) – дефекты, вызывающие сбои системы или потерю данных, блокирующие дальнейшее тестирование.
- Высокой важности (Major) – дефекты, существенно влияющие на функциональность или производительность.
- Средней важности (Normal) – дефекты, не критичные, но ухудшающие качество продукта.
- Низкой важности (Minor) – косметические дефекты или незначительные несоответствия.

По типу обнаруженного дефекта
- Функциональные – нарушения в выполнении функций системы.
- Нефункциональные – связанные с производительностью, безопасностью, удобством использования, совместимостью и др.
- Графические – ошибки отображения интерфейса.
- Логические – ошибки в логике работы программы.
- Интеграционные – проблемы при взаимодействии различных компонентов системы.
- Технические – ошибки установки, конфигурации или окружения.

По стадии жизненного цикла разработки
- Требования – несоответствия требованиям заказчика или спецификациям.
- Проектирование – ошибки в архитектуре или проектных решениях.
- Реализация – ошибки в коде или реализации функций.
- Тестирование – дефекты, выявленные на этапе тестирования.
- Эксплуатация – дефекты, обнаруженные после внедрения системы.

По происхождению
- Человеческие ошибки – опечатки, неправильное понимание требований.
- Технические сбои – сбои оборудования или программных средств.
- Процедурные ошибки – неправильное выполнение процессов разработки или тестирования.

По мнению Канера С. не рекомендуется использовать определение программной ошибки как расхождение между программой и ее спецификацией потому, что расхождение между программой и ее спецификацией является ошибкой тогда, и только тогда, когда спецификация существует и она правильна.
Поэтому следующие два определения более точны.
- Если программа не делает того, чего пользователь от нее вполне обоснованно ожидает, значит, налицо программная ошибка (Myers, 1976).
- Не существует ни абсолютного определения ошибок, ни точного критерия наличия их в программе. Можно лишь сказать, насколько программа не справляется со своей задачей, – это исключительно субъективная характеристика (Beizer, 1984).

Рассмотрим предложенные Канером С категории ошибок в программном обеспечении.
- Ошибки пользовательского интерфейса.
- Функциональные недостатки имеют место, если программа не делает того, что должна, выполняет одну из своих функций плохо или не полностью.
- Взаимодействие программы с пользователем – насколько сложно пользователю разобраться в том, как работать с программой?
- Организация программы – насколько легко потеряться в вашей программе?
- Пропущенные команды – Чего в программе не хватает?
- Производительность – в интерактивном программном обеспечении очень важна скорость.
- Выходные данные – получаете ли пользователь то, что хотел?
- Обработка ошибок. Процедуры обработки ошибок – это очень важная часть программы.
- Ошибки, связанные с обработкой граничных условий. Любой аспект работы программы, к которому применимы понятия больше или меньше, раньше или позже, первый или последний, короче или длиннее, обязательно должен быть проверен на границах диапазона. Внутри диапазонов программа обычно работает прекрасно, а вот на их границах порой случаются самые неожиданные отклонения.
- Ошибки вычислений. Например, одной из самых распространенных среди математических ошибок являются ошибка округления.
- Начальное и последующие состояния. Бывает, что при выполнении какой-либо функции программы сбой происходит только однажды – при самом первом обращении к этой функции.

- Ошибки управления потоком. Если по логике программы вслед за первым действием должно быть выполнено второе, а она выполняет третье, значит, в управлении потоком допущена ошибка.
- Ошибки передачи или интерпретации данных – это когда изменения, внесенные одной из частей программы, могут потеряться или достичь не всех частей системы, где они важны.
- Ситуация гонок. Классическая ситуация гонок описывается так. Предположим, в системе ожидаются два события, А и Б. Первым может произойти любое из них. Но если первым произойдет событие А, выполнение программы продолжится, а если первым наступит событие Б, то в работе программы произойдет сбой. Программист полагал, что первым всегда должно быть событие А, и не ожидал, что Б может выиграть гонки.
- Перегрузки. Программа может не справляться с повышенными нагрузками. Например, она может не выдерживать интенсивной и длительной эксплуатации или не справляться со слишком большими объемами данных.
- Аппаратное обеспечение – например, программы могут посылать устройствам неверные данные, игнорировать их сообщения об ошибках, пытаться использовать устройства, которые заняты или вообще отсутствуют.
- Контроль версий. Бывает, что старые ошибки вдруг всплывают снова из-за того, что программа скомпонована с устаревшей версией одной из подпрограмм.
- Документация. Сама по себе документация не является программным обеспечением, но все же это часть программного продукта. И если она плохо написана, пользователь может подумать, что и сама программа не намного лучше.
- Ошибки тестирования. Если программист допускает по полторы ошибки на каждую строку программного кода, то, сколько их допускает тестировщик на каждый тест?

Таким образом, классификация дефектов, обнаруженных при тестировании, помогает систематизировать и управлять выявленными ошибками для их эффективного устранения и предотвращения в будущем. Эта классификация помогает определить приоритеты исправления ошибок и улучшить процессы разработки и тестирования.

6.3. Метрики оценивания программных продуктов

Документ, описывающий и регламентирующий перечень работ по тестированию, а также соответствующие техники и подходы, стратегию, области ответственности, ресурсы, расписание и ключевые даты называется тест-план (test plan). Один из разделов тест-плана – это метрики.

Метрики (metrics). Числовые характеристики показателей качества, способы их оценки, формулы и т.д. На этот раздел, как правило, формируется множество ссылок из других разделов тест-плана.

Метрика (metric) – числовая характеристика показателя качества. Может включать описание способов оценки и анализа результата.

Метрики могут быть как прямыми, которые не требуют вычислений, так и расчетными – вычисляются по формуле. Типичные примеры прямых метрик – количество разработанных тест-кейсов, количество найденных дефектов и т.д. В расчетных метриках могут использоваться как совершенно тривиальные, так и довольно сложные формулы.

В тестировании существует большое количество общепринятых метрик, многие из которых могут быть собраны автоматически с использованием инструментальных средств управления проектами. Например:
- процентное отношение (не) выполненных тест-кейсов ко всем имеющимся;
- процентный показатель успешного прохождения тест-кейсов;
- процентный показатель заблокированных тест-кейсов;
- плотность распределения дефектов;
- эффективность устранения дефектов;
- распределение дефектов по важности и срочности;

и т.д

Как правило, при формировании отчетности нас будет интересовать не только текущее значение метрики, но и ее динамика во времени, которую очень удобно изображать графически, что тоже могут выполнять автоматически многие средства управления проектами.

Покрытие (coverage) – процентное выражение степени, в которой исследуемый элемент (coverage item) затронут соответствующим набором тест-кейсов.

Самыми простыми представителями метрик покрытия можно считать:
- Метрику покрытия требований – требование считается «покрытым», если на него ссылается хотя бы один тест-кейс.
- Метрику плотности покрытия требований – учитывается, сколько тест-кейсов ссылается на несколько требований.
- Метрику покрытия классов эквивалентности – анализируется, сколько классов эквивалентности затронуто тест-кейсами.
- Метрику покрытия граничных условий – анализируется, сколько значений из группы граничных условий затронуто тест-кейсами.

Метрики покрытия кода модульными тест-кейсами. Таких метрик очень много, но вся их суть сводится к выявлению некоей характеристики кода, например, количество строк, ветвей, путей, условий и т.д. и определению, какой процент представителей этой характеристики покрыт тест-кейсами.

6.4. Критерии завершения тестирования

Критерии завершения тестирования – это совокупность условий, при выполнении которых считается, что тестирование можно считать завершенным. Они помогают определить, достигнуты ли цели тестирования и можно ли перейти к следующему этапу разработки или выпуска продукта.

Основные критерии завершения тестирования включают:
- Достижение запланированного объема тестирования – это значит, что выполнены все запланированные тестовые сценарии и случаи, проверки всех функциональных и нефункциональных требований.
- Достижение установленных уровней качества – это когда доля обнаруженных дефектов ниже допустимого порога и нет критических или блокирующих дефектов, ожидающих исправления.
- Выполнение плановых сроков – завершение тестирования в рамках запланированного графика.
- Устранение основных дефектов – все критические и важные дефекты устранены или зафиксированы как неустранимые по причине их низкой приоритетности.
- Оценка покрытия тестами. Достигнуто достаточное покрытие кода, требований или сценариев, например, 90% покрытие по коду или требованиям.
- Статус качества продукта – уровень дефектности и качество соответствуют установленным стандартам или требованиям заказчика.
- Документальное завершение – это когда подготовлены все необходимые отчеты, документация по результатам тестирования.
- Одобрение заинтересованных сторон, т.е. получено согласие от менеджеров, заказчиков или других заинтересованных лиц о завершении тестирования.

Эти критерии могут варьироваться в зависимости от проекта, методологии разработки, например, Agile, Waterfall, требований заказчика и специфики продукта. Важно заранее согласовать критерии завершения тестирования с командой и заинтересованными сторонами для обеспечения прозрачности и понимание целей процесса.

Лабораторная работа. Создание и оформление отчетности о результатах тестирования.

Выполните функционального тестирования веб-приложения, используя разработанные ранее тесты. Создайте и оформите отчет о результатах тестирования согласно рекомендациям.

7. МЕТОДЫ ТЕСТИРОВАНИЯ

7.1. Классификация методов тестирования. Обзор методов тестирования

Тестирование можно классифицировать по очень большому количеству признаков. Приведем упрощенный вариант классификации тестирования.

По доступу к коду и архитектуре приложения

- Метод белого ящика – доступ к коду есть.
- Метод черного ящика – доступа к коду нет.

По степени автоматизации

- Ручное тестирование – тест-кейсы выполняет человек.
- Автоматизированное тестирование – тест-кейсы частично или полностью выполняет специальное инструментальное средство.

По уровню детализации приложения (по уровню тестирования)

- Модульное (компонентное) тестирование – проверяются отдельные небольшие части приложения.
- Интеграционное тестирование – проверяется взаимодействие между несколькими частями приложения.
- Системное тестирование – приложение проверяется как единое целое.

По принципам работы с приложением

- Позитивное тестирование – все действия с приложением выполняются строго по инструкции без никаких недопустимых действий, некорректных данных и т.д. Можно образно сказать, что приложение исследуется в «тепличных условиях».
- Негативное тестирование – в работе с приложением выполняются некорректные операции и используются данные, потенциально приводящие к ошибкам, например, деление на ноль.

По привлечению конечных пользователей

- Альфа-тестирование.
- Бета-тестирование.
- Гамма-тестирование.

7.2. Методы функционального тестирования, структурные, направленного поиска ошибок, основанные на типе программного обеспечения и на использовании

Функциональное тестирование (functional testing) – вид тестирования, направленный на проверку корректности работы функциональности приложения (корректность реализации функциональных требований).

Методы функционального тестирования, структурные

Часто функциональное тестирование ассоциируют с тестированием по методу черного ящика, однако и по методу белого ящика вполне можно проверять корректность реализации функциональности.

С другой стороны нефункциональное тестирование (non-functional testing) – вид тестирования, направленный на проверку нефункциональных особенностей приложения, корректность реализации нефункциональных требований, таких как удобство использования, совместимость, производительность, безопасность и т.д.

Одним из методов функционального тестирования может быть ручное тестирование.

Ручное тестирование (manual testing) – тестирование, в котором тест-кейсы выполняются человеком вручную без использования средств автоматизации. Несмотря на то что это звучит очень просто, от тестировщика в те или иные моменты времени требуются такие качества, как терпеливость, наблюдательность, креативность, умение ставить нестандартные эксперименты, а также умение видеть и понимать, что происходит «внутри системы», т.е. как внешние воздействия на приложение трансформируются в его внутренние процессы.

Ручное тестирование может применяться для тестирования различного уровня функциональности приложения и используется в следующих видах тестирования:

- дымовое тестирование,
- тестирование критического пути.

Дымовое тестирование (smoke test, intake test, build verifi cation test) направлено на проверку самой главной, самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной саму идею использования приложения (или иного объекта, подвергаемого дымовому тестированию).

Тестирование критического пути (critical path test) направлено на исследование функциональности, используемой типичными пользователями в типичной повседневной деятельности. Как видно из определения в сноске к англоязычной версии термина, сама идея позаимствована из управления проектами и трансформирована в контексте тестирования в следующую: существует большинство пользователей, которые чаще всего используют некое подмножество функций приложения. Именно эти функции и нужно проверить, как только мы убедились, что приложение «в принципе работает» (дымовой тест прошел успешно). Если по каким-то причинам приложение не выполняет эти функции или выполняет их некорректно, очень многие пользователи не смогут достичь множества своих целей.

Метод направленного поиска ошибок

Замечено, что некоторые люди по своим качествам оказываются прекрасными специалистами по тестированию программ. Они обладают умением «выискивать» ошибки без привлечения какой-либо методологии тестировании. Объяснятся это тем, что человек, обладающий практическим опытом, часто подсознательно применяет метод проектирования тестов, называемым педположением об ошибке. Данный метод тестирование можно назвать свободным тестированием.

Свободное, интуитивное тестирование – полностью неформализованный подход, в котором не предполагается использования ни тест-кейсов, ни чек-листов, ни сценариев – тестировщик полностью опирается на свой профессионализм и интуицию для спонтанного выполнения с приложением действий, которые, как он считает, могут обнаружить ошибку. Этот вид тестирования используется редко и исключительно как дополнение к полностью или частично формализованному тестированию в случаях, когда для исследования некоторого аспекта поведения приложения при отсутствии тест-кейсов.

Процедуру для метода предположения об ошибке описать трудно, так как он в значительной степени является интуитивным. Основная идее его заключается в том, чтобы перечислить в некотором списке возможные ошибки или ситуации, в которых они могут появляться, а затем на основе этого списка написать тесты.

Пример. Рассмотрим тестирование подпрограммы сортировки. В данном случае необходимо исследовать следующие ситуации.

1. Сортируемый список пуст.
2. Сортируемый список содержит только одно значение.
3. Все записи в сортируемом списке имеют одно и то же значение.
4. Список уже отсортирован.

Методы тестирования, основанные на типе программного обеспечения и на использовании

Существующее программное обеспечение может быть представлено как настольное приложение, мобильное приложение и веб-приложение. Так же может быть встроенной частью устройства, например, банкомат, стиральная машина, телевизор и т.д. В данном случае можно говорить о природе приложения и соответственно классифицировать тестирование по природе приложения.

Данный вид классификации является искусственным, поскольку «внутри» речь будет идти об одних и тех же видах тестирования, отличающихся в данном контексте лишь концентрацией на соответствующих функциях и особенностях приложения, использованием специфических инструментов и отдельных техник.

Тестирование веб-приложений сопряжено с интенсивной деятельностью в области тестирования совместимости, в особенности – кросс-браузерного тестирования, тестирования производительности, автоматизации тестирования с использованием широкого спектра инструментальных средств.
Тестирование мобильных приложений (mobile applications testing) также требует повышенного внимания к тестированию совместимости, оптимизации производительности (в том числе клиентской части с точки зрения снижения энергопотребления), автоматизации тестирования с применением эмуляторов мобильных устройств.
Тестирование настольных приложений (desktop applications testing) является самым классическим среди всех перечисленных в данной классификации, и его особенности зависят от предметной области приложения, нюансов архитектуры, ключевых показателей качества и т.д.
Эту классификацию можно продолжать, например, можно отдельно рассматривать тестирование консольных приложений (console applications testing) и приложений с графическим интерфейсом (GUI-applications testing), серверных приложений (server applications testing) и клиентских приложений (client applications testing) и т.д.

7.4. Сущность методов эквивалентного разбиения и анализа граничных значений.

Процедура разбиения входного пространства на категории

Класс эквивалентности (equivalence class) – набор данных, обрабатываемых одинаковым образом и приводящих к одинаковому результату.
Граничное условие (border condition, boundary condition) – значение, находящееся на границе классов эквивалентности.

Метод эквивалентного разбиения

Тестирование программного обеспечения ограничивается использованием небольшого подмножества всех возможных входных данных. Сущность метода эквивалентного разбиения входных значений заключается в выборе подходящего подмножества входных значений, которое позволит с наивысшей вероятностью обнаружить большинство ошибок в программном обеспечении.
Эквивалентное разбиение составляет основу методологии тестирования по принципу черного ящика. Метод черного ящика (black box testing, closed box testing, specifi cation-based testing) – у тестировщика либо нет доступа к внутренней структуре и коду приложения, либо недостаточно знаний для их понимания, либо он сознательно не обращается к ним в процессе тестирования.
Классы эквивалентности выделяются путем выбора каждого входного условия, значения и разбиением его на две или более групп. Различают два типа классов эквивалентности: правильные классы эквивалентности, представляющие правильные, допустимые входные данные программы, и неправильные классы эквивалентности, представляющие все другие возможные состояния условий, т.е. ошибочные входные значения, недопустимые.
Правильные и неправильные классы эквивалентности лежат в основе классификации видов и направлений тестирования по принципам работы программного обеспечения: позитивное и негативное тестирование.
Позитивное тестирование (positive testing) направлено на исследование приложения в ситуации, когда все действия выполняются строго по инструкции, без каких бы то ни было ошибок, отклонений, ввода неверных данных и т.д. Если позитивные тест-кейсы завершаются ошибками, это тревожный признак – приложение работает неверно даже в идеальных условиях и можно предположить, что в неидеальных условиях оно работает еще хуже. Для ускорения тестирования несколько позитивных тест-кейсов можно объединять, например, перед отправкой заполнить все поля формы верными значениями, – иногда это может усложнить диагностику ошибки, но существенная экономия времени компенсирует этот риск.
Негативное тестирование (negative testing, invalid testing) – направлено на исследование работы приложения в ситуациях, когда с ним выполняются некорректные операции или используются данные, потенциально приводящие к ошибкам. Поскольку в реальной жизни таких ситуаций значительно больше, поскольку пользователи допускают ошибки, злоумышленники осознанно «ломают» приложение, в среде работы приложения возникают проблемы и т.д., негативных тест-кейсов оказывается значительно больше, чем позитивных в разы или даже на порядки. В отличие от позитивных, негативные тест-кейсы не стоит объединять, т.к. подобное решение может привести к неверной трактовке поведения приложения и пропуску, не обнаружению дефектов.
Процесс выделения классов эквивалентности представляет собой в значительной степени эвристический, творческий процесс. Не смотря на творческий процесс, существует ряд правил.
1. Если входное условие описывает область значений, например, «целое данное может принимать значения от 1 до 999», то определяются один правильный класс эквивалентности ($1 \leq$ значение целого данного ≤ 999) и два неправильных (значение целого данного < 1 и значение целого данного > 999).
2. Если входное условие описывает число значений, например, «в автомобиле могут ехать от одного до шести человек», то определяются один правильный класс эквивалентности и два неправильных (ни одного и более шести человек).
3. Если входное условие описывает множество входных значений и есть основание полагать, что каждое значение программа трактует особо, например, «известны способы передвижения на Автобусе, Грузовике, Такси, Пешком или Мотоцикле», то определяется один правильный класс эквивалентности для каждого значения и один неправильный класс эквивалентности, например, «на Прицепе».
4. Если входное условие описывает ситуацию «должно быть», например, «первым символом идентификатора должна быть буква», то определяется один правильный класс эквивалентности (первый символ – буква) и один неправильный (первый символ – не буква).
5. Если есть любое основание считать, что различные элементы класса эквивалентности трактуются программой неодинаково, то данный класс эквивалентности разбивается на меньшие классы эквивалентности.
Выбор значений для неправильного класса эквивалентности зачастую затруднен при тестировании по методу черного ящика.
Если существует доступ к коду приложения, то всегда можно посмотреть, как реализованы проверки входных данных на допустимые и недопустимые значения, что позволяет подобрать значения входных данных для негативного тестирования.
Метод белого ящика – у тестировщика есть доступ к внутренней структуре и коду приложения, а также есть достаточно знаний для понимания увиденного. Выделяют даже сопутствующую тестированию по методу белого ящика глобальную технику – тестирование на основе дизайна. Для более глубокого изучения сути метода белого ящика рекомендуется ознакомиться с техниками исследования потока управления или потока данных, использования диаграмм состояний. Некоторые авторы склонны жестко связывать этот метод со статическим тестированием, но ничто не мешает тестировщику запустить код на выполнение и при этом периодически обращаться к самому коду (а модульное тестирование и вовсе предполагает запуск кода на исполнение и при этом работу именно с кодом, а не с «приложением целиком»).

Анализ граничных значений

Граничные условия – это ситуации, возникающие непосредственно на, выше или ниже границ входных классов эквивалентности. Анализ граничных значений отличается от эквивалентного разбиения.

- Выбор любого элемента в классе эквивалентности в качестве представительного при анализе граничных значений осуществляется таим образом, чтобы проверить тестом каждую границу этого класса.
- При разработке тестов рассматривают не только входные условия (пространство входов), но и пространство результатов (т.е. выходные классы эквивалентности).

Анализ граничных значений требует эвристического, творческого подхода и специализации в рассматриваемой области, проблеме и в значительной мере основывается на способностях человеческого интеллекта. Тем не менее существуют некоторые общие правила.

1. Построить тесты для границ области и тесты с неправильными входными данными для ситуаций незначительного выхода за границы области, если входное условие описывает область значений.

Пример. Если правильная область входных значений есть интервал от -1.0 до 1.0, то выбрать в качестве граничных значений -1.001, -1.0, -0.999, 0.999, 1.0, 1,001.

2. Построить тесты для минимального и максимального значений условий и тесты, большие и меньшие этих значений, если входное условие удовлетворяет дискретному ряду значений.

Пример. Если входной файл может содержать от 1 до 255 записей, то получить тесты для 0, 1, 2, 254, 255, 256 записей.

3. Использовать правило 1 для каждого выходного условия. Заметим, что важно проверить границы пространства результатов, поскольку не всегда границы входных областей представляют такой же набор условий, как и границы выходных областей. Не всегда также можно построить результат вне выходной области, но, тем не менее стоит рассмотреть эту возможность.

Пример. Если программа вычисляет ежемесячный расход и если минимум расхода составляет 0.00 руб., а максимум – 1165.25 руб., то построить тесты, которые вызывают расходы с 0.00 руб. и 1165.25 руб. Кроме того, построить, если это возможно, тесты, которые вызывают отрицательный расход и расход больше 1165.25 руб.

4. Использовать правило 2 для каждого выходного условия.

Пример. Если система информационного поиска отображает на экране терминала наиболее релевантные записи в зависимости от входного запроса, но никак не более четырех записей, то следует построить тесты, такие, чтобы программа отображала нуль, один и четыре записи, и тест, который мог бы вызвать выполнение программы с ошибочным отображением пяти записей.

5. Если вход и выход программы есть упорядоченное множество.

Пример. Линейный список, таблица, то сосредоточить внимание на первом и последнем элементах этого множества.

6. Попробовать свои силы в поиске другие граничных условий.

Таким образом, граничные условия могут быть едва уловимы и, следовательно, определение их связано с большими трудностями.

СПИСОК ИСТОЧНИКОВ

- Гагарина, Л. Г. Технология разработки программного обеспечения: учебное пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Сидорова-Виснадул ; под ред. Л. Г. Гагариной.- М: ФОРУМ: ИНФРА-М, 2023.- 400 с.
- Куликов, С. С. Тестирование программного обеспечения : учеб. пособие / С. С. Куликов [и др.]. – Минск : БГУИР, 2019. – 276 с.
- Лаврищева, Е. М. Программная инженерия и технологии программирования сложных систем : учебник для вузов / Е. М. Лаврищева.- 2-е изд., испр. и доп.- М .: Издательство Юрайт, 2019.- 432 с.
- Стандартный глоссарий терминов, используемых в тестировании программного обеспечения [Электронный ресурс] / Режим доступа: https://www.gasq.Org/files/content/gasq/downloads/certification/ISTQB/Glossary/ISTQB_Glossary_Russian_v2_0.pdf
- Майерс, Г. Искусство тестирования программ / Г. Майерс, Т. Баджетт, К. Сандлер.- 3-е изд.- М: СПб: Диалектика, 2019- 271 с.
- Бек, К. Экстремальное программирование: разработка через тестирование : практическое руководство / К. Бек.- Санкт-Петербург : Питер, 2021.- 224 с.
- Котляров, В. П. Основы тестирования программного обеспечения / В.П. Котляров, Т. В. Коликова.- М: Интернет-Университет информационных технологий, Бином. Лаборатория знаний.- 2-е изд.- М.: Интуит, 2016.- 348 с.
- Старолетов, С. М. Основы тестирования и верификации программного обеспечения / С. М. Старолетов- СПБ: Лань, 2018.- 344 с.
- Уиттакер, Дж. Как тестируют в Google / Дж. Уиттакер, Дж. Арбон.- М: Питер, 2014.- 320 с.