ТЕМА 4.7 ТЕСТИРОВАНИЕ И ОТЛАДКА ОКОННЫХ ПРИЛОЖЕНИЙ

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

На протяжении десятилетий развития разработки ПО к вопросам тестирования и обеспечения качества подходили очень и очень по-разному. Можно выделить несколько основных «эпох тестирования».

В 50–60-х годах прошлого века процесс тестирования был предельно формализован, отделён от процесса непосредственной разработки ПО и «математизирован». Фактически тестирование представляло собой скорее отладку программ (debugging3). Существовала концепция т.н. «исчерпывающего тестирования (exhaustive testing4)» — 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, а также 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 testing5), а затем — строго наоборот: как процесс доказательства неработоспособности программы в некоторых заданных условиях (negative testing6). Это внутреннее противоречие не только не исчезло со временем, но и в наши дни многими авторами совершенно справедливо отмечается как две взаимодополняющие цели тестирования.

Отметим, что «процесс доказательства неработоспособности программы» ценится чуть больше, т.к. не позволяет закрывать глаза на обнаруженные проблемы.

Итак, ещё раз самое важное, что тестирование «приобрело» в 70-е годы:

• тестирование позволяет удостовериться, что программа соответствует требованиям;

• тестирование позволяет определить условия, при которых программа ведёт себя некорректно.

В 80-х годах произошло ключевое изменение места тестирования в разработке ПО: вместо одной из финальных стадий создания проекта тестирование стало применяться на протяжении всего цикла разработки (software lifecycle7), что позволило в очень многих случаях не только быстро обнаруживать и устранять проблемы, но даже предсказывать и предотвращать их появление. В этот же период времени отмечено бурное развитие и формализация методологий тестирования и появление первых элементарных попыток автоматизировать тестирование.

В 90-х годах произошёл переход от тестирования как такового к более всеобъемлющему процессу, который называется «обеспечение качества (quality assurance8)», охватывает весь цикл разработки ПО 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 привёл 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 всё новых и новых путей, методологий, техник и подходов к обеспечению качества. Серьёзное 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 её способность предоставить конечному пользователю возможность эффективно решать свои задачи.

О современном этапе развития тестирования мы будем говорить на протяжении всего остального материала. Если же отметить вкратце основные характеристики, то получится примерно такой список: 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).

Виды и направления тестирования

Упрощённая классификация тестирования

Тестирование можно классифицировать по очень большому количеству признаков, и практически в каждой серьёзной книге о тестировании автор показывает свой (безусловно имеющий право на существование) взгляд на этот вопрос.

Тестирование можно классифицировать:

Рисунок 2.3.а — Упрощённая классификация тестирования

• По запуску кода на исполнение:

о Статическое тестирование — без запуска.

о Динамическое тестирование — с запуском.

• По доступу к коду и архитектуре приложения:

о Метод белого ящика — доступ к коду есть.

о Метод чёрного ящика — доступа к коду нет.

о Метод серого ящика — к части кода доступ есть, к части — нет.

• По степени автоматизации:

о Ручное тестирование — тест-кейсы выполняет человек.

о Автоматизированное тестирование — тест-кейсы частично или полностью выполняет специальное инструментальное средство.

• По уровню детализации приложения (по уровню тестирования):

о Модульное (компонентное) тестирование — проверяются отдельные небольшие части приложения.

о Интеграционное тестирование — проверяется взаимодействие между несколькими частями приложения.

о Системное тестирование — приложение проверяется как единое целое.

• По (убыванию) степени важности тестируемых функций (по уровню функционального тестирования):

о Дымовое тестирование — проверка самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной саму идею использования приложения.

о Тестирование критического пути — проверка функциональности, используемой типичными пользователями в типичной повседневной деятельности.

о Расширенное тестирование — проверка всей (остальной) функциональности, заявленной в требованиях.

• По принципам работы с приложением:

o 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».

o Негативное тестирование — в работе с приложением выполняются (некорректные) операции и используются данные, потенциально приводящие к ошибкам (классика жанра — деление на ноль).

Классификация по запуску кода на исполнение

Далеко не всякое тестирование предполагает взаимодействие с работающим приложением. Потому в рамках данной классификации выделяют:

• Статическое тестирование (static testing) — тестирование без запуска кода на исполнение. В рамках этого подхода тестированию могут подвергаться:

o Документы (требования, тест-кейсы, описания архитектуры приложения, схемы баз данных и т.д.).

o Графические прототипы (например, эскизы пользовательского интерфейса).

o Код приложения (что часто выполняется самими программистами в рамках аудита кода (code review), являющегося специфической вариацией взаимного просмотра в применении к исходному коду). Код приложения также можно проверять с использованием техник тестирования на основе структур кода.

o Параметры (настройки) среды исполнения приложения.

o Подготовленные тестовые данные.

• Динамическое тестирование (dynamic testing) — тестирование с запуском кода на исполнение. Запускаться на исполнение может как код всего приложения целиком (системное тестирование), так и код нескольких взаимосвязанных частей (интеграционное тестирование), отдельных частей (модульное или компонентное тестирование) и даже отдельные участки кода. Основная идея этого вида тестирования состоит в том, что проверяется реальное поведение (части) приложения.

Классификация по доступу к коду и архитектуре приложения

• 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. Выделяют даже сопутствующую тестированию по методу белого ящика глобальную технику — тестирование на основе дизайна (design-based testing). Для более глубокого изучения сути метода белого ящика рекомендуется ознакомиться с техниками исследования потока управления или потока данных, использования диаграмм состояний. Некоторые авторы склонны жёстко 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»).

• Метод чёрного ящика (black box testing, closed box testing, specificationbased 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. В рамках тестирования по методу чёрного ящика основной информацией для создания тест-кейсов выступает документация (особенно — требования (requirements-based testing)) и общий здравый смысл (для случаев, когда поведение приложения в некоторой ситуации не регламентировано явно; иногда это называют «тестированием на основе неявных требований», но канонического определения у этого подхода нет).

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

Методы белого и чёрного ящика не являются конкурирующими или взаимоисключающими — напротив, они гармонично дополняют друг друга, компенсируя таким образом имеющиеся недостатки.

	Преимущества	Недостатки
Метод белого ящика	• Показывает скрытые проблемы и упрощает их диагностику. • Допускает достаточно простую автоматизацию тест-кейсов и их выполнение на самых ранних стадиях развития проекта. • Обладает развитой системой метрик, сбор и анализ которых легко автоматизируется. • Стимулирует разработчиков к написанию качественного кода. • Многие техники этого метода являются проверенными, хорошо себя зарекомендовавшими решениями, базирующимися на строгом техническом подходе.	• Не может выполняться тестировщиками, не обладающими достаточными знаниями в области программирования. • Тестирование сфокусировано на реализованной функциональности, что повышает вероятность пропуска нереализованных требований. • Поведение приложения исследуется в отрыве от реальной среды выполнения и не учитывает её влияние. • Поведение приложения исследуется в отрыве от реальных пользовательских сценариев[146].
Метод чёрного ящика	• Тестировщик не обязан обладать (глубокими) знаниями в области программирования. • Поведение приложения исследуется в контексте реальной среды выполнения и учитывает её влияние. • Поведение приложения исследуется в контексте реальных пользовательских сценариев[146]. • Тест-кейсы можно создавать уже на стадии появления стабильных требований. • Процесс создания тест-кейсов позволяет выявить дефекты в требованиях. • Допускает создание тест-кейсов, которые можно многократно использовать на разных проектах.	• Возможно повторение части тест-кейсов, уже выполненных разработчиками. • Высока вероятность того, что часть возможных вариантов поведения приложения останется непротестированной. • Для разработки высокоэффективных тест-кейсов необходима качественная документация. • Диагностика обнаруженных дефектов более сложна в сравнении с техниками метода белого ящика. • В связи с широким выбором техник и подходов затрудняется планирование и оценка трудозатрат. • В случае автоматизации могут потребоваться сложные дорогостоящие инструментальные средства.
Метод серого ящика	Сочетает преимущества и недостатки методов белого и чёрного ящика.	

Классификация по степени автоматизации

• Ручное тестирование (manual testing) — тестирование, в котором тесткейсы 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.

• Автоматизированное тестирование (automated testing, test automation) — набор техник, подходов и инструментальных средств, позволяющий исключить человека из выполнения некоторых задач в процессе тестирования. Тест-кейсы частично или полностью выполняет специальное инструментальное средство, однако разработка тест-кейсов, подготовка данных, оценка результатов выполнения, написания отчётов об обнаруженных дефектах — всё это и многое другое по-прежнему делает человек.

Преимущества	Недостатки
• Скорость выполнения тест-кейсов может в разы и на порядки превосходить возможности человека. • Отсутствие влияния человеческого фактора в процессе выполнения тест-кейсов (усталости, невнимательности и т.д.). • Минимизация затрат при многократном выполнении тест-кейсов (участие человека здесь требуется лишь эпизодически). • Способность средств автоматизации выполнить тест-кейсы, в принципе непосильные для человека в силу своей сложности, скорости или иных факторов. • Способность средств автоматизации собирать, сохранять, анализировать, агрегировать и представлять в удобной для восприятия человеком форме колоссальные объёмы данных. • Способность средств автоматизации выполнять низкоуровневые действия с приложением, операционной системой, каналами передачи данных и т.д.	• Необходим высококвалифицированный персонал в силу того факта, что автоматизация — это «проект внутри проекта» (со своими требованиями, планами, кодом и т.д.) • Высокие затраты на сложные средства автоматизации, разработку и сопровождение кода тест-кейсов. • Автоматизация требует более тщательного планирования и управления рисками, т.к. в противном случае проекту может быть нанесён серьёзный ущерб. • Средств автоматизации крайне много, что усложняет проблему выбора того или иного средства и может повлечь за собой финансовые затраты (и риски), необходимость обучения персонала (или поиска специалистов). • В случае ощутимого изменения требований, смены технологического домена, переработки интерфейсов (как пользовательских, так и программных) многие тест-кейсы становятся безнадёжно устаревшими и требуют создания заново.

Если же выразить все преимущества и недостатки автоматизации тестирования одной фразой, то получается, что автоматизация позволяет ощутимо увеличить тестовое покрытие (test coverage), но при этом столь же ощутимо увеличивает риски.

Классификация по уровню детализации приложения (по уровню тестирования)

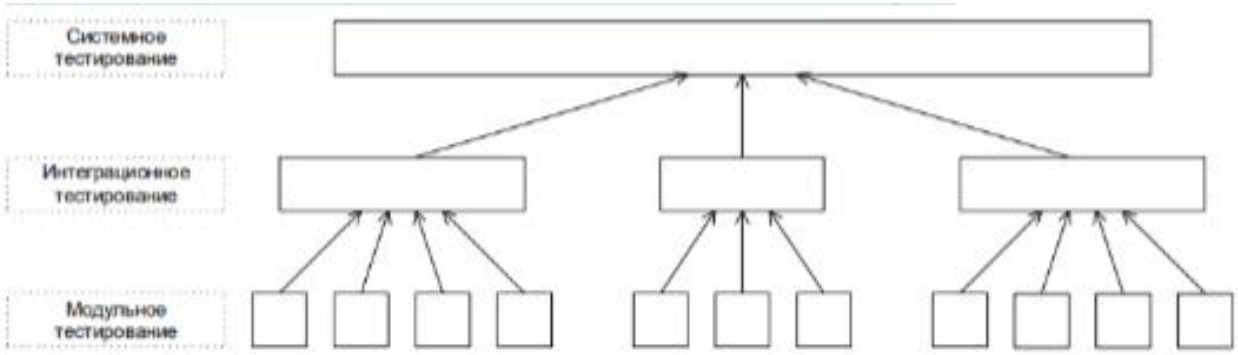
• Модульное (компонентное) тестирование (unit testing, module testing, component testing) направлено на проверку отдельных небольших частей приложения, которые (как правило) можно исследовать изолированно от других подобных частей. При выполнении данного тестирования могут проверяться отдельные функции 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 (integration testing, component integration testing, pairwise integration testing, system integration testing, incremental testing, interface testing, thread 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 мы работаем с очень качественными отдельными компонентами, «на стыке» их взаимодействия часто возникают проблемы. Именно эти проблемы и выявляет интеграционное 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.

С классификацией по уровню детализации приложения связан интересный печальный факт: если предыдущая стадия обнаружила проблемы, то на следующей стадии эти проблемы точно нанесут удар по качеству; если же предыдущая стадия не обнаружила проблем, это ещё никоим образом не защищает нас от проблем на следующей стадии.

Для лучшего запоминания степень детализации в модульном, интеграционном и системном тестировании показана схематично на рисунке:



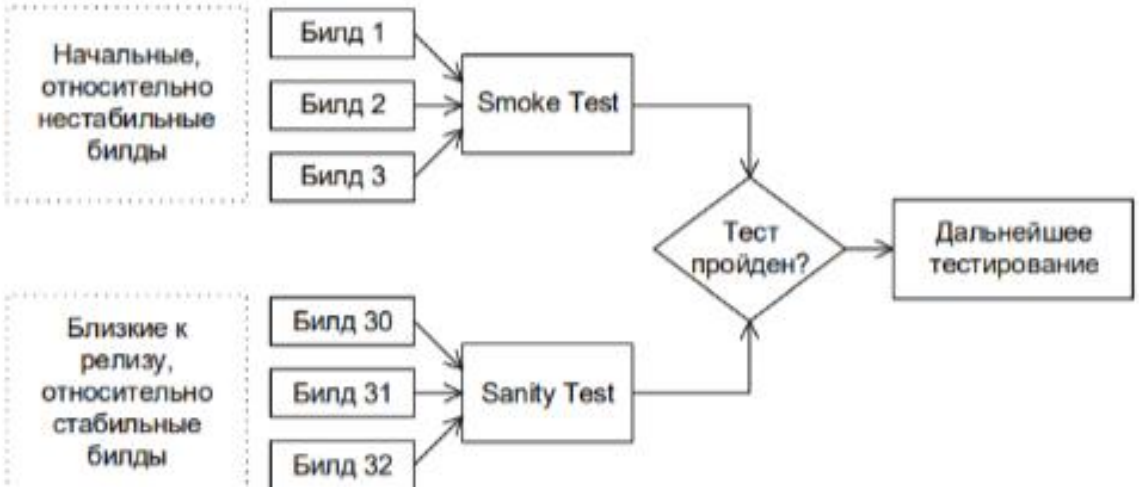
Классификация по (убыванию) степени важности тестируемых функций (по уровню функционального тестирования)

В некоторых источниках эту разновидность классификации также называют «по глубине тестирования».

• Дымовое тестирование (smoke test, intake test, build verification 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 (не)целесообразности 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 %.

Очень часто можно услышать вопрос о том, чем «smoke test» отличается от «sanity test». В глоссарии ISTQB сказано просто: «sanity test: See smoke test». Но некоторые авторы утверждают, что разница есть и может быть выражена следующей схемой



• 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 прошёл 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 всё равно достаточно высоко (как правило, порядка 70–80–90 % — в зависимости от сути проекта).

• 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 тесткейсы этого уровня могут затронуть даже самые низкоприоритетные требования.

Ещё 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 (иногда можно увидеть даже значения в диапазоне 30–50 %, т.к. подавляющее большинство найденных здесь дефектов не представляет угрозы для успешного использования приложения большинством пользователей).

Классификация по принципам работы с приложением

• 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 ещё 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 и/или используются данные, потенциально приводящие к ошибкам (классика жанра — деление на ноль). Поскольку в реальной жизни таких ситуаций значительно больше (пользователи допускают ошибки, злоумышленники осознанно «ломают» приложение, в среде работы приложения возникают проблемы и т.д.), негативных тест-кейсов оказывается значительно больше, чем позитивных (иногда — 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 (необнаружению) дефектов.

Классификация 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 (web-applications testing) 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 очень долго. 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.

Классификация по фокусировке на уровне архитектуры приложения

Данный вид классификации, как и предыдущий, также является искусственным и отражает лишь концентрацию внимания на отдельной части приложения.

• Тестирование уровня представления (presentation tier testing) сконцентрировано на той части приложения, которая отвечает за взаимодействие с «внешним миром» (как пользователями, так и другими приложениями). Здесь исследуются вопросы удобства использования, скорости отклика интерфейса, совместимости с браузерами, корректности работы интерфейсов.

• Тестирование уровня бизнес-логики (business logic tier testing) отвечает за проверку основного набора функций приложения и строится на базе ключевых требований к приложению, бизнес-правил и общей проверки функциональности.

• Тестирование уровня данных (data tier testing) сконцентрировано на той части приложения, которая отвечает за хранение и некоторую обработку данных (чаще всего — в базе данных или ином хранилище). Здесь особый интерес представляет тестирование данных, проверка соблюдения бизнес-правил, тестирование производительности.

Классификация по привлечению конечных пользователей

Все три перечисленных ниже вида тестирования относятся к операционному тестированию.

• Альфа-тестирование (alpha testing) выполняется внутри организации-разработчика с возможным частичным привлечением конечных пользователей. Может являться формой внутреннего приёмочного тестирования. В некоторых источниках отмечается, что это тестирование должно проводиться без привлечения команды разработчиков, но другие источники не выдвигают такого требования. Суть этого вида вкратце: продукт уже можно периодически показывать внешним пользователям, но он ещё достаточно «сырой», потому основное тестирование выполняется организацией-разработчиком.

• Бета-тестирование (beta testing) выполняется вне организации-разработчика с активным привлечением конечных пользователей/заказчиков. Может являться формой внешнего приёмочного тестирования. Суть этого вида вкратце: продукт уже можно открыто

показывать внешним пользователям, он уже достаточно стабилен, но проблемы всё ещё могут быть, и для их выявления нужна обратная связь от реальных пользователей.

• Гамма-тестирование (gamma testing) — финальная стадия тестирования перед выпуском продукта, направленная на исправление незначительных дефектов, обнаруженных в бета-тестировании. Как правило, также выполняется с максимальным привлечением конечных пользователей/заказчиков. Может являться формой внешнего приёмочного тестирования. Суть этого вида вкратце: продукт уже почти готов, и сейчас обратная связь от реальных пользователей используется для устранения последних недоработок.

Классификация по степени формализации

• Тестирование на основе тест-кейсов (scripted testing, test case based testing) — формализованный подход, в котором тестирование производится на основе заранее подготовленных тест-кейсов, наборов тест-кейсов и иной документации. Это самый распространённый способ тестирования, который также позволяет достичь максимальной полноты исследования приложения за счёт строгой систематизации процесса, удобства применения метрик и широкого набора выработанных за десятилетия и проверенных на практике рекомендаций.

• Исследовательское тестирование (exploratory testing) — частично формализованный подход, в рамках которого тестировщик выполняет работу с приложением по выбранному сценарию, который, в свою очередь, дорабатывается в процессе выполнения с целью более полного исследования приложения. Ключевым фактором успеха при выполнении исследовательского тестирования является именно работа по сценарию, а не выполнение разрозненных бездумных операций. Существует даже специальный сценарный подход, называемый сессионным тестированием (session-based testing). В качестве альтернативы сценариям при выборе действий с приложением иногда могут использоваться чек-листы, и тогда этот вид тестирования называют тестированием на основе чек-листов (checklist-based testing).

• Свободное (интуитивное) тестирование (ad hoc testing) — полностью неформализованный подход, в котором не предполагается использования ни тест-кейсов, ни чек-листов, ни сценариев — тестировщик полностью опирается на свой профессионализм и интуицию (experience-based testing) 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 (пока?) нет тест-кейсов.

Классификация по целям и задачам

• Позитивное тестирование (рассмотрено ранее).

• Негативное тестирование (рассмотрено ранее).

• Функциональное тестирование (functional testing) — вид тестирования, направленный на проверку корректности работы функциональности приложения (корректность реализации функциональных требований). Часто функциональное тестирование ассоциируют с тестированием по методу чёрного ящика, однако и по методу белого ящика вполне можно проверять корректность реализации функциональности.

• 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.

• Инсталляционное тестирование (installation testing, installability testing) — тестирование, направленное на выявление дефектов, влияющих на протекание стадии инсталляции (установки) приложения. В общем случае такое тестирование проверяет множество сценариев и аспектов работы инсталлятора в таких ситуациях, как:

о новая среда исполнения, в которой приложение ранее не было инсталлировано;

о обновление существующей версии («апгрейд»);

о изменение текущей версии на более старую («даунгрейд»);

о повторная установка приложения с целью устранения возникших проблем («переинсталляция»);

о повторный запуск инсталляции после ошибки, приведшей к невозможности продолжения инсталляции;

о удаление приложения;

о установка нового приложения из семейства приложений; о автоматическая инсталляция без участия пользователя.

• 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. Фредерик Брукс в своей книге «Мифический человеко-месяц» писал: «Фундаментальная проблема при сопровождении программ состоит в том, что исправление одной ошибки с большой вероятностью (20–50 %) влечёт появление новой». Потому регрессионное тестирование является неотъемлемым инструментом обеспечения качества и активно используется практически в любом проекте.

• Повторное тестирование (re-testing, confirmation testing) — выполнение тест-кейсов, которые ранее обнаружили дефекты, с целью подтверждения устранения дефектов. Фактически этот вид тестирования сводится к действиям на финальной стадии жизненного цикла отчёта о дефекте, направленным на то, чтобы перевести дефект в состояние «проверен» и «закрыт».

• Приёмочное 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). Можно выделить следующие подвиды приёмочного тестирования (хотя упоминают их крайне редко, ограничиваясь в основном общим термином «приёмочное тестирование»):

о Производственное приёмочное тестирование (factory acceptance testing) — выполняемое проектной командой исследование полноты и качества реализации приложения с точки зрения его готовности к передаче заказчику. Этот вид тестирования часто рассматривается как синоним альфа-тестирования.

о Операционное приёмочное тестирование (operational acceptance testing, production acceptance testing) — операционное тестирование, выполняемое с точки зрения выполнения инсталляции, потребления приложением ресурсов, совместимости с программной и аппаратной платформой и т.д.

о Итоговое приёмочное тестирование (site acceptance testing) — тестирование конечными пользователями (представителями заказчика) приложения в реальных условиях эксплуатации с целью вынесения решения о том, требует ли приложение доработок или может быть принято в эксплуатацию в текущем виде.

• Операционное тестирование (operational testing) — тестирование, проводимое в реальной или приближенной к реальной операционной среде (operational environment), включающей операционную систему, системы управления базами данных, серверы приложений, веб-серверы, аппаратное обеспечение и т.д.

• Тестирование удобства использования (usability testing) — тестирование, направленное на исследование того, насколько конечному пользователю понятно, как работать с продуктом (understandability, learnability, operability), а также на то, насколько ему нравится использовать продукт (attractiveness). И это не оговорка — очень часто успех продукта зависит именно от эмоций, которые он вызывает у пользователей. Для эффективного проведения этого вида тестирования требуется реализовать достаточно серьёзные исследования с привлечением конечных пользователей, проведением маркетинговых исследований и т.д.

• Тестирование доступности (accessibility testing) — тестирование, направленное на исследование пригодности продукта к использованию людьми с ограниченными возможностями (слабым зрением и т.д.).

• Тестирование интерфейса (interface testing) — тестирование, направленное на проверку интерфейсов приложения или его компонентов. По определению ISTQB-глоссария этот вид тестирования относится к интеграционному тестированию, и это вполне справедливо для таких его вариаций как тестирование интерфейса прикладного программирования (API testing) и интерфейса командной строки (CLI testing), хотя последнее может выступать и как разновидность тестирования пользовательского интерфейса, если через командную строку с приложением взаимодействует пользователь, а не другое приложение. Однако многие источники предлагают включить в состав тестирования интерфейса и тестирование непосредственно интерфейса пользователя (GUI testing).

• Тестирование безопасности (security testing) — тестирование, направленное на проверку способности приложения противостоять злонамеренным попыткам получения доступа к данным или функциям, права на доступ к которым у злоумышленника нет.

• Тестирование интернационализации (internationalization testing, in testing, globalization testing, localizability testing) — тестирование, направленное на проверку готовности продукта к работе с использованием различных языков и с учётом различных национальных и культурных особенностей. Этот вид тестирования не подразумевает проверки качества соответствующей адаптации (этим занимается тестирование локализации), оно сфокусировано именно на проверке возможности такой адаптации (например: что будет, если открыть файл с иероглифом в имени; как будет работать интерфейс, если всё перевести на японский; может ли приложение искать данные в тексте на корейском и т.д.).

• Тестирование локализации (localization testing) — тестирование, направленное на проверку корректности и качества адаптации продукта к использованию на том или ином языке с учётом национальных и культурных особенностей. Это тестирование следует за тестированием интернационализации и проверяет корректность перевода и адаптации продукта, а не готовность продукта к таким действиям.

• Тестирование совместимости (compatibility testing, interoperability testing) — тестирование, направленное на проверку способности приложения работать в указанном окружении. Здесь, например, может проверяться:

o Совместимость с аппаратной платформой, операционной системой и сетевой инфраструктурой (конфигурационное тестирование, configuration testing).

o Совместимость с браузерами и их версиями (кросс-браузерное тестирование, cross-browser testing).

o Совместимость с мобильными устройствами (mobile testing).

• Тестирование данных (data quality testing) и баз данных (database integrity testing) — два близких по смыслу вида тестирования, направленных на исследование таких характеристик данных, как полнота, непротиворечивость, целостность, структурированность и т.д. В контексте баз данных исследованию может подвергаться адекватность модели предметной области, способность модели обеспечивать целостность и консистентность данных, корректность работы триггеров, хранимых процедур и т.д.

• Тестирование использования ресурсов (resource utilization testing, efficiency testing, storage testing) — совокупность видов тестирования, проверяющих эффективность использования приложением доступных ему ресурсов и зависимость результатов работы приложения от количества доступных ему ресурсов. Часто эти виды тестирования прямо или косвенно примыкают к техникам тестирования производительности.

• Сравнительное тестирование (comparison testing) — тестирование, направленное на сравнительный анализ преимуществ и недостатков разрабатываемого продукта по отношению к его основным конкурентам.

• Демонстрационное тестирование (qualification testing) — формальный процесс демонстрации заказчику продукта с целью подтверждения, что продукт соответствует всем заявленным требованиям. В отличие от приёмочного тестирования этот процесс более

строгий и всеобъемлющий, но может проводиться и на промежуточных стадиях разработки продукта.

• Исчерпывающее тестирование (exhaustive testing) — тестирование приложения со всеми возможными комбинациями всех возможных входных данных во всех возможных условиях выполнения. Для сколь бы то ни было сложной системы нереализуемо, но может применяться для проверки отдельных крайне простых компонентов.

• Тестирование надёжности (reliability testing) — тестирование способности приложения выполнять свои функции в заданных условиях на протяжении заданного времени или заданного количества операций.

• Тестирование восстанавливаемости (recoverability testing) — тестирование способности приложения восстанавливать свои функции и заданный уровень производительности, а также восстанавливать данные в случае возникновения критической ситуации, приводящей к временной (частичной) утрате работоспособности приложения.

• Тестирование отказоустойчивости (failover testing) — тестирование, заключающееся в эмуляции или реальном создании критических ситуаций с целью проверки способности приложения задействовать соответствующие механизмы, предотвращающие нарушение работоспособности, производительности и повреждения данных.

• Тестирование производительности (performance testing) — исследование показателей скорости реакции приложения на внешние воздействия при различной по характеру и интенсивности нагрузке. В рамках тестирования производительности выделяют следующие подвиды:

o Нагрузочное тестирование (load testing, capacity testing) — исследование способности приложения сохранять заданные показатели качества при нагрузке в допустимых пределах и некотором превышении этих пределов (определение «запаса прочности»).

o Тестирование масштабируемости (scalability testing) — исследование способности приложения увеличивать показатели производительности в соответствии с увеличением количества доступных приложению ресурсов.

o Объёмное тестирование (volume testing) — исследование производительности приложения при обработке различных (как правило, больших) объёмов данных.

o Стрессовое тестирование (stress testing) — исследование поведения приложения при нештатных изменениях нагрузки, значительно превышающих расчётный уровень, или в ситуациях недоступности значительной части необходимых приложению ресурсов. Стрессовое тестирование может выполняться и вне контекста нагрузочного тестирования: тогда оно, как правило, называется «тестированием на разрушение» (destructive testing) и представляет собой крайнюю форму негативного тестирования.

o Конкурентное тестирование (concurrency testing) — исследование поведения приложения в ситуации, когда ему приходится обрабатывать большое количество одновременно поступающих запросов, что вызывает конкуренцию между запросами за ресурсы (базу данных, память, канал передачи данных, дисковую подсистему и т.д.). Иногда под конкурентным тестированием понимают также исследование работы многопоточных приложений и корректность синхронизации действий, производимых в разных потоках.

Классификация по техникам и подходам

• Позитивное тестирование (рассмотрено ранее).
• Негативное тестирование (рассмотрено ранее).
• Тестирование на основе опыта тестировщика, сценариев, чек-листов:
o Исследовательское тестирование
o Свободное (интуитивное) тестирование (рассмотрено ранее).
• Классификация по степени вмешательства в работу приложения:

o Инвазивное тестирование (intrusive testing) — тестирование, выполнение которого может повлиять на функционирование приложения в силу работы инструментов тестирования (например, будут искажены показатели производительности) или в силу вмешательства (level of intrusion) в сам код приложения (например, для анализа работы

приложения было добавлено дополнительное протоколирование, включён вывод отладочной информации и т.д.).

o Неинвазивное тестирование (nonintrusive testing) — тестирование, выполнение которого незаметно для приложения и не влияет на процесс его обычной работы.

• Классификация по техникам автоматизации:

o Тестирование под управлением данными (data-driven testing) — способ разработки автоматизированных тест-кейсов, в котором входные данные и ожидаемые результаты выносятся за пределы тесткейса и хранятся вне его — в файле, базе данных и т.д.

o Тестирование под управлением ключевыми словами (keyworddriven 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 тесткейса, которая описывается ключевыми словами (командами).

o 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.

• Классификация на основе (знания) источников ошибок:

o Тестирование предугадыванием ошибок (error guessing) — техника тестирования, в которой тесты разрабатываются на основе опыта тестировщика и его знаний о том, какие дефекты типичны для тех или иных компонентов или областей функциональности приложения. Может комбинироваться с техникой т.н. «ошибкоориентированного» тестирования (failure-directed testing), в котором новые тесты строятся на основе информации

o ранее обнаруженных в приложении проблемах.

o Эвристическая оценка (heuristic evaluation) — техника тестирования удобства использования, направленная на поиск проблем в интерфейсе пользователя, представляющих собой отклонение от общепринятых норм.

o Мутационное тестирование (mutation testing) — техника тестирования, в которой сравнивается поведение нескольких версий одного и того же компонента, причём часть таких версий может быть специально разработана с добавлением ошибок (что позволяет оценить эффективность тест-кейсов — качественные тесты обнаружат эти специально добавленные ошибки). Может комбинироваться со следующим в этом списке видом тестирования (тестированием добавлением ошибок).

o Тестирование добавлением ошибок (error seeding) — техника тестирования, в которой в приложение специально добавляются заранее известные, специально продуманные ошибки с целью мониторинга их обнаружения и устранения и, таким образом, формирования более точной оценки показателей процесса тестирования. Может комбинироваться с предыдущим в этом списке видом тестирования (мутационным тестированием).

• Классификация на основе выбора входных данных:

o Тестирование на основе классов эквивалентности (equivalence partitioning) — техника тестирования, направленная на сокращение количества разрабатываемых и выполняемых тест-кейсов при сохранении достаточного тестового покрытия. Суть техники состоит в выявлении наборов эквивалентных тест-кейсов (каждый из которых проверяет одно и то же поведение приложения) и выборе из таких наборов небольшого подмножества тест-кейсов, с наибольшей вероятностью обнаруживающих проблему.

o Тестирование на основе граничных условий (boundary value analysis) — инструментальная техника тестирования на основе классов эквивалентности, позволяющая выявить специфические значения исследуемых параметров, относящиеся к границам классов эквивалентности. Эта техника значительно упрощает выявление наборов эквивалентных тест-кейсов и выбор таких тест-кейсов, которые обнаружат проблему с наибольшей вероятностью.

o Доменное тестирование (domain analysis, domain testing) — техника тестирования на основе классов эквивалентности и граничных условий, позволяющая эффективно создавать тест-кейсы, затрагивающие несколько параметров (переменных) одновременно (в том числе с учётом взаимозависимости этих параметров). Данная техника также описывает подходы к

выбору минимального множества показательных тест-кейсов из всего набора возможных тест-кейсов.

o Попарное тестирование (pairwise testing) — техника тестирования, в которой тест-кейсы строятся по принципу проверки пар значений параметров (переменных) вместо того, чтобы пытаться проверить все возможные комбинации всех значений всех параметров. Эта техника является частным случаем N-комбинационного тестирования (n-wise testing) и позволяет существенно сократить трудозатраты на тестирование (а иногда и вовсе сделать возможным тестирование в случае, когда количество «всех комбинаций всех значений всех параметров» измеряется миллиардами).

o Тестирование на основе ортогональных массивов (orthogonal array testing) — инструментальная техника попарного и Nкомбинационного тестирования, основанная на использовании т.н. «ортогональных массивов» (двумерных массивов, обладающих следующим свойством: если взять две любые колонки такого массива, то получившийся «подмассив» будет содержать все возможные попарные комбинации значений, представленных в исходном массиве).

• Классификация на основе среды выполнения:

o Тестирование в процессе разработки (development testing) — тестирование, выполняемое непосредственно в процессе разработки приложения и/или в среде выполнения, отличной от среды реального использования приложения. Как правило, выполняется самими разработчиками.

o Операционное тестирование (рассмотрено ранее).

• Тестирование на основе кода (code based testing). В различных источниках эту технику называют по-разному (чаще всего — тестированием на основе структур, причём некоторые авторы смешивают в один набор тестирование по потоку управления и по потоку данных, а некоторые строго разделяют эти стратегии). Подвиды этой техники также организуют в разные комбинации, но наиболее универсально их можно классифицировать так:

o Тестирование по потоку управления (control flow testing) — семейство техник тестирования, в которых тест-кейсы разрабатываются с целью активации и проверки выполнения различных последовательностей событий, которые определяются посредством анализа исходного кода приложения.

o Тестирование по потоку данных (data-flow testing) — семейство техник тестирования, основанных на выборе отдельных путей из потока управления с целью исследования событий, связанных с изменением состояния переменных.

o Тестирование по диаграмме или таблице состояний (state transition testing) — техника тестирования, в которой тест-кейсы разрабатываются для проверки переходов приложения из одного состояния в другое. Состояния могут быть описаны диаграммой состояний (state diagram) или таблицей состояний (state table).

Иногда эту технику тестирования также называют «тестированием по принципу конечного автомата» (finite state machine testing). Важным преимуществом этой техники является возможность применения в ней теории конечных автоматов (которая хорошо формализована), а также возможность использования автоматизации для генерации комбинаций входных данных.

o Инспекция (аудит) кода (code review, code inspection) — семейство техник повышения качества кода за счёт того, что в процессе создания или совершенствования кода участвуют несколько человек. Степень формализации аудита кода может варьироваться от достаточно беглого просмотра до тщательной формальной инспекции. В отличие от техник статического анализа кода (по потоку управления и потоку данных) аудит кода также улучшает такие его характеристики, как понятность, поддерживаемость, соответствие соглашениям об оформлении и т.д. Аудит кода выполняется в основном самими программистами.

• Тестирование на основе структур кода (structure-based techniques) предполагает возможность исследования логики выполнения кода в зависимости от различных ситуаций и включает в себя:

o Тестирование на основе выражений (statement testing) — техника тестирования (по методу белого ящика), в которой проверяется корректность (и сам факт) выполнения отдельных выражений в коде.

o Тестирование на основе ветвей (branch testing) — техника тестирования (по методу белого ящика), в которой проверяется выполнение отдельных ветвей кода (под ветвью понимается атомарная часть кода, выполнение которой происходит или не происходит в зависимости от истинности или ложности некоторого условия).

o Тестирование на основе условий (condition testing) — техника тестирования (по методу белого ящика), в которой проверяется выполнение отдельных условий (условием считается выражение, которое может быть вычислено до значения «истина» или «ложь»).

o Тестирование на основе комбинаций условий (multiple condition testing) — техника тестирования (по методу белого ящика), в которой проверяется выполнение сложных (составных) условий.

o Тестирование на основе отдельных условий, порождающих ветвление («решающих условий») (modified condition decision coverage testing) — техника тестирования (по методу белого ящика), в которой проверяется выполнение таких отдельных условий в составе сложных условий, которые в одиночку определяют результат вычисления всего сложного условия.

o Тестирование на основе решений (decision testing) — техника тестирования (по методу белого ящика), в которой проверяется выполнение сложных ветвлений (с двумя и более возможными вариантами). Несмотря на то, что «два варианта» сюда также подходит, формально такую ситуацию стоит отнести к тестированию на основе условий.

o Тестирование на основе путей (path testing) — техника тестирования (по методу белого ящика), в которой проверяется выполнение всех или некоторых специально выбранных путей в коде приложения.

• Тестирование на основе (моделей) поведения приложения (application behavior/model-based testing):

o Тестирование по таблице принятия решений (decision table testing) — техника тестирования (по методу чёрного ящика), в которой тест-кейсы разрабатываются на основе т.н. таблицы принятия решений, в которой отражены входные данные (и их комбинации) и воздействия на приложение, а также соответствующие им выходные данные и реакции приложения. o Тестирование по диаграмме или таблице состояний.

o Тестирование по спецификациям (specification-based testing, black box testing).

o Тестирование по моделям поведения приложения (model-based testing) — техника тестирования, в которой исследование приложения (и разработка тест-кейсов) строится на некой модели: таблице принятия решений, таблице или диаграмме состояний, пользовательских сценариев, модели нагрузки и т.д.

o Тестирование на основе вариантов использования (use case testing) — техника тестирования (по методу чёрного ящика), в которой тест-кейсы разрабатываются на основе вариантов использования. Варианты использования выступают в основном источником информации для шагов тест-кейса, в то время как наборы входных данных удобно разрабатывать с помощью техник выбора входных данных. В общем случае источником информации для разработки тест-кейсов в этой технике могут выступать не только варианты использования, но и другие пользовательские требования в любом их виде. В случае если методология разработки проекта подразумевает использование пользовательских историй, этот вид тестирования может быть заменён тестированием на основе пользовательских историй (user story testing).

o Параллельное тестирование (parallel testing) — техника тестирования, в которой поведение нового (или модифицированного) приложения сравнивается с поведением эталонного приложения (предположительно работающего верно). Термин «параллельное тестирование» также может использоваться для обозначения способа проведения тестирования, когда несколько тестировщиков или систем автоматизации выполняют работу одновременно, т.е. параллельно. Очень редко (и не совсем верно) под параллельным тестированием понимают мутационное тестирование.

o Тестирование на основе случайных данных (random testing) — техника тестирования (по методу чёрного ящика), в которой входные данные, действия или даже сами тест-кейсы выбираются на основе (псевдо)случайных значений так, чтобы соответствовать операционному профилю (operational profile) — подмножеству действий, соответствующих некоей ситуации или сценарию работы с приложением. Не стоит путать этот вид тестирования с т.н. «обезьяньим тестированием» (monkey testing).

o A/B-тестирование (A/B testing, split testing264) — техника тестирования, в которой исследуется влияние на результат выполнения операции изменения одного из входных параметров. Однако куда чаще можно встретить трактовку A/B-тестирования как технику тестирования удобства использования, в которой пользователям случайным образом предлагаются разные варианты элементов интерфейса, после чего оценивается разница в реакции пользователей.

Классификация по моменту выполнения (хронологии)

Несмотря на многочисленные попытки создать единую хронологию тестирования, предпринятые многими авторами, по-прежнему можно смело утверждать, что общепринятого решения, которое в равной степени подходило бы для любой методологии управления проектами, любого отдельного проекта и любой его стадии, просто не существует.

Если попытаться описать хронологию тестирования одной общей фразой, то можно сказать, что происходит постепенное наращивание сложности самих тесткейсов и сложности логики их выбора.

• Общая универсальная логика последовательности тестирования состоит в том, чтобы начинать исследование каждой задачи с простых позитивных тест-кейсов, к которым постепенно добавлять негативные (но тоже достаточно простые). Лишь после того, как наиболее типичные ситуации покрыты простыми тест-кейсами, следует переходить к более сложным (опять же, начиная с позитивных). Такой подход — не догма, но к нему стоит прислушаться, т.к. углубление на начальных этапах в негативные (к тому же — сложные) тест-кейсы может привести к ситуации, в которой приложение отлично справляется с кучей неприятностей, но не работает на элементарных повседневных задачах. Ещё раз суть универсальной последовательности:

1) простое позитивное тестирование;
2) простое негативное тестирование;
3) сложное позитивное тестирование;
4) сложное негативное тестирование.

• Последовательность тестирования, построенная по иерархии компонентов:

o Восходящее тестирование (bottom-up testing) — инкрементальный подход к интеграционному тестированию, в котором в первую очередь тестируются низкоуровневые компоненты, после чего процесс переходит на всё более и более высокоуровневые компоненты.

o Нисходящее тестирование (top-down testing) — инкрементальный подход к интеграционному тестированию, в котором в первую очередь тестируются высокоуровневые компоненты, после чего процесс переходит на всё более и более низкоуровневые компоненты.

o Гибридное тестирование (hybrid testing) — комбинация восходящего и нисходящего тестирования, позволяющая упростить и ускорить получение результатов оценки приложения.

• Последовательность тестирования, построенная по концентрации внимания на требованиях и их составляющих:

1) Тестирование требований, которое может варьироваться от беглой оценки в стиле «всё ли нам понятно» до весьма формальных подходов, в любом случае первично по отношению к тестированию того, как эти требования реализованы.

2) Тестирование реализации функциональных составляющих требований логично проводить до тестирования реализации нефункциональных составляющих, т.к. если что-то просто не работает, то проверять производительность, безопасность, удобство и прочие нефункциональные составляющие бессмысленно, а чаще всего и вовсе невозможно.

3) Тестирование реализации нефункциональных составляющих требований часто становится логическим завершением проверки того, как реализовано то или иное требование.

• Типичные общие сценарии используются в том случае, когда не существует явных предпосылок к реализации иной стратегии. Такие сценарии могут видоизменяться и комбинироваться (например, весь «типичный общий сценарий 1» можно повторять на всех шагах «типичного общего сценария 2»).

o Типичный общий сценарий 1:

1) Дымовое тестирование.

2) Тестирование критического пути.

3) Расширенное тестирование.

o Типичный общий сценарий 2:

1) Модульное тестирование.

2) Интеграционное тестирование.

3) Системное тестирование.

o Типичный общий сценарий 3:

1) Альфа-тестирование.

2) Бета-тестирование.

3) Гамма-тестирование