# Лекция 2.2.1. Детальная классификация тестирования

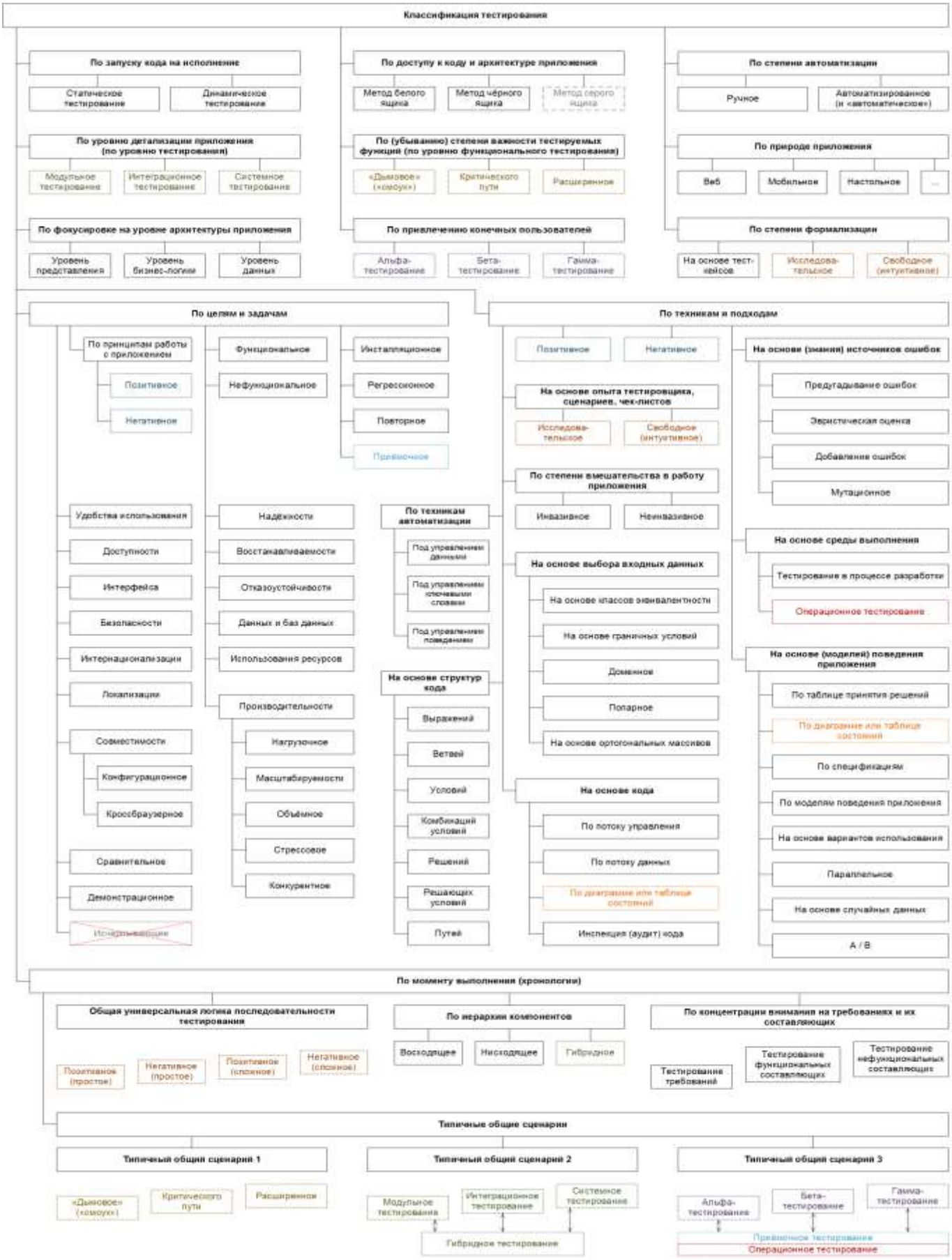


*Рисунок – Классификация тестирования*

Теперь мы рассмотрим классификацию тестирования максимально подробно. Сейчас вы приступите к изучению одного из самых сложных разделов нашего обучения.

Давайте коротко пройдемся по всем видам тестирования, о которых мы говорили во время первого урока второго модуля.

### *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:
  - Документы (требования, тест-кейсы, описания архитектуры приложения, схемы баз данных и т.д.).
  - Графические прототипы (например, эскизы пользовательского интерфейса).
  - 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.
  - Параметры (настройки) среды исполнения приложения.
  - 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 степени автоматизации*

- 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.
- 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. Некоторые авторы говорят отдельно о «полуавтоматизированном» тестировании как варианте ручного с частичным использованием средств автоматизации и отдельно об «автоматизированном» тестировании (относя туда области тестирования, в которых компьютер выполняет ощутимо большой процент задач). Но т.к. без участия человека всё равно не обходится ни один из этих видов тестирования, не станем усложнять набор терминов и ограничимся одним понятием «автоматизированное тестирование». У автоматизированного тестирования есть много как сильных, так и слабых сторон Давайте проговорим про преимущества и недостатки автоматизированного тестирования.

- 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)***

Внимание! Возможна путаница, вызванная тем, что единого общепринятого набора классификаций не существует, и две из них имеют очень схожие названия:

- «По уровню детализации приложения» = «по уровню тестирования».
- «По (убыванию) степени важности тестируемых функций» = «по уровню функционального 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.
- Интеграционное тестирование (integration) 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.

Внимание! Очень распространённая проблема! Из-за особенности перевода на русский язык под термином «приёмочное тестирование» часто может пониматься как «smoke test», так и «acceptance test», которые изначально не имеют между собою ничего общего. Возможно, в том числе поэтому многие тестировщики почти не используют русский перевод «дымовое тестирование», а так и говорят — «смоук-тест». 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.

- Тестирование критического пути (critical path test) направлено на исследование функциональности, используемой типичными пользователями в типичной повседневной деятельности.
- 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. Ещё одним направлением исследования в рамках данного тестирования являются нетипичные, маловероятные, экзотические случаи и сценарии использования функций и свойств приложения, затронутых на предыдущих уровнях.

К сожалению, часто можно встретить мнение, что дымовое тестирование, тестирование критического пути и расширенное тестирование напрямую связаны с позитивным тестированием и негативным тестированием, и негативное появляется только на уровне тестирования критического пути. Это не так. Как позитивные, так и негативные тесты могут (а иногда и обязаны) встречаться на всех перечисленных уровнях. Например, деление на ноль в калькуляторе явно должно относиться к дымовому тестированию, хотя это яркий пример негативного тест-кейса.

- 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) — 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. Если вы не знакомы с понятием многоуровневой архитектуры приложений, не волнуйтесь, мы пройдем это в последующих уроках.

### 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). Дополнительную информацию об исследовательском тестировании можно получить из статьи Джеймса Баха «Что такое исследовательское тестирование?»
- 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. Ни в коем случае не стоит путать исследовательское и свободное тестирование. Это разные техники исследования приложения с разной степенью формализации, разными задачами и областями применения.