2.3. Виды и направления тестирования

2.3.1. 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.

Соответствующий материал достаточно объёмен и сложен, а глубокое понимание каждого пункта в классификации требует определённого опыта, потому мы разделим данную тему на две: сейчас мы рассмотрим самый простой, минимальный набор информации, необходимый начинающему тестировщику, а в следующей главе приведём подробную классификацию.

Используйте нижеприведённый список как очень краткую «шпаргалку для запоминания». Итак, тестирование можно классифицировать:



Рисунок 2.3.a — Упрощённая классификация тестирования

- По запуску кода на исполнение:
  - Статическое тестирование — без запуска.
  - Динамическое тестирование — с запуском.
- По доступу к коду и архитектуре приложения:
  - Метод белого ящика — доступ к коду есть.
  - Метод чёрного ящика — доступа к коду нет.
  - Метод серого ящика — к части кода доступ есть, к части — 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 (убыванию) степени важности тестируемых функций (по уровню функционального тестирования):
  - Дымовое тестирование (обязательно изучите этимологию термина — хотя бы в Википедии[108]) — проверка самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной саму идею использования приложения.

[108] «Smoke test», Wikipedia [http://en.wikipedia.org/wiki/Smoke_testing_(electrical)]

    - Тестирование критического пути — проверка функциональности, используемой типичными пользователями в типичной повседневной деятельности.
    - Расширенное тестирование — проверка всей (остальной) функциональности, заявленной в требованиях.

- 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 (классика жанра — деление на ноль).



Внимание! Очень частая ошибка! Негативные тесты НЕ предполагают возникновения в приложении ошибки. Напротив — они предполагают, что верно работающее приложение даже в критической ситуации поведёт себя правильным образом (в примере с делением на ноль, например, отобразит сообщение «Делить на ноль запрещено»).

2.3.2. Подробная классификация тестирования

2.3.2.1. Схема классификации тестирования

Теперь мы рассмотрим классификацию тестирования максимально подробно. Настоятельно рекомендуется прочесть не только текст этой главы, но и все дополнительные источники, на которые будут приведены ссылки.

На рисунках 2.3.b и 2.3.c приведена схема, на которой все способы классификации показаны одновременно. Многие авторы, создававшие подобные классификации[109], использовали интеллект-карты, однако такая техника не позволяет в полной мере отразить тот факт, что способы классификации пересекаются (т.е. некоторые виды тестирования можно отнести к разным способам классификации). На рисунках 2.3.b и 2.3.c самые яркие случаи таких пересечений отмечены цветом (см. полноразмерный электронный вид рисунка[111]) и границей блоков в виде набора точек. Если вы видите на схеме подобный блок — ищите одноимённый где-то в другом виде классификации.



Настоятельно рекомендуется в дополнение к материалу этой главы прочесть прекрасную статью «Классификация видов тестирования»[109], а также классическую книгу Ли Коупленда «Практическое руководство по разработке тестов» (Lee Copeland, «A Practitioner's Guide to Software Test Design»).

Зачем вообще нужна классификация тестирования? Она позволяет упорядочить знания и значительно ускоряет процессы планирования тестирования и разработки тест-кейсов, а также позволяет оптимизировать трудозатраты за счёт того, что тестировщику не приходится изобретать очередной велосипед.

При этом ничто не мешает создавать собственные классификации — как вообще придуманные с нуля, так и представляющие собой комбинации и модификации представленных ниже классификаций.



Если вас интересует некая «эталонная классификация», то… её не существует. Можно сказать, что в материалах[110] ISTQB приведён наиболее обобщённый и общепринятый взгляд на этот вопрос, но и там нет единой схемы, которая объединяла бы все варианты классификации.

Так что, если вас просят рассказать о классификации тестирования, стоит уточнить, согласно какому автору или источнику спрашивающий ожидает услышать ваш ответ.

Сейчас вы приступите к изучению одного из самых сложных разделов этой книги. Если вы уже имеете достаточный опыт в тестировании, можете отталкиваться от схемы, чтобы систематизировать и расширить свои знания. Если вы только начинаете заниматься тестированием, рекомендуется сначала прочитать текст, следующий за схемой.



По поводу схем, которые вы сейчас увидите на рисунках 2.3.b и 2.3.c, часто поступают вопросы, почему функциональное и нефункциональное тестирование не связано с соответствующими подвидами. Тому есть две причины:
1) Несмотря на то что те или иные виды тестирования принято причислять к функциональному или нефункциональному тестированию, в них всё

[109] «Классификация видов тестирования» [http://habrahabr.ru/company/npo-comp/blog/223833/]
[110] International Software Testing Qualifications Board, Downloads. [http://www.istqb.org/downloads.html]

	равно присутствуют обе составляющие (как функциональная, так и нефункциональная), пусть и в разных пропорциях. 2) Схема превратилась бы в непроглядную паутину линий. Потому было решено оставить рисунки 2.3.b и 2.3.c в том виде, в каком они представлены на следующих двух страницах. Полноразмерный вариант этих рисунков можно скачать здесь[111].

Итак, тестирование можно классифицировать…

[111] Полноразмерный вариант рисунков 2.3.b [http://svyatoslav.biz/wp-pics/software_testing_classification_ru.png] и 2.3.c [http://svyatoslav.biz/wp-pics/software_testing_classification_en.png]

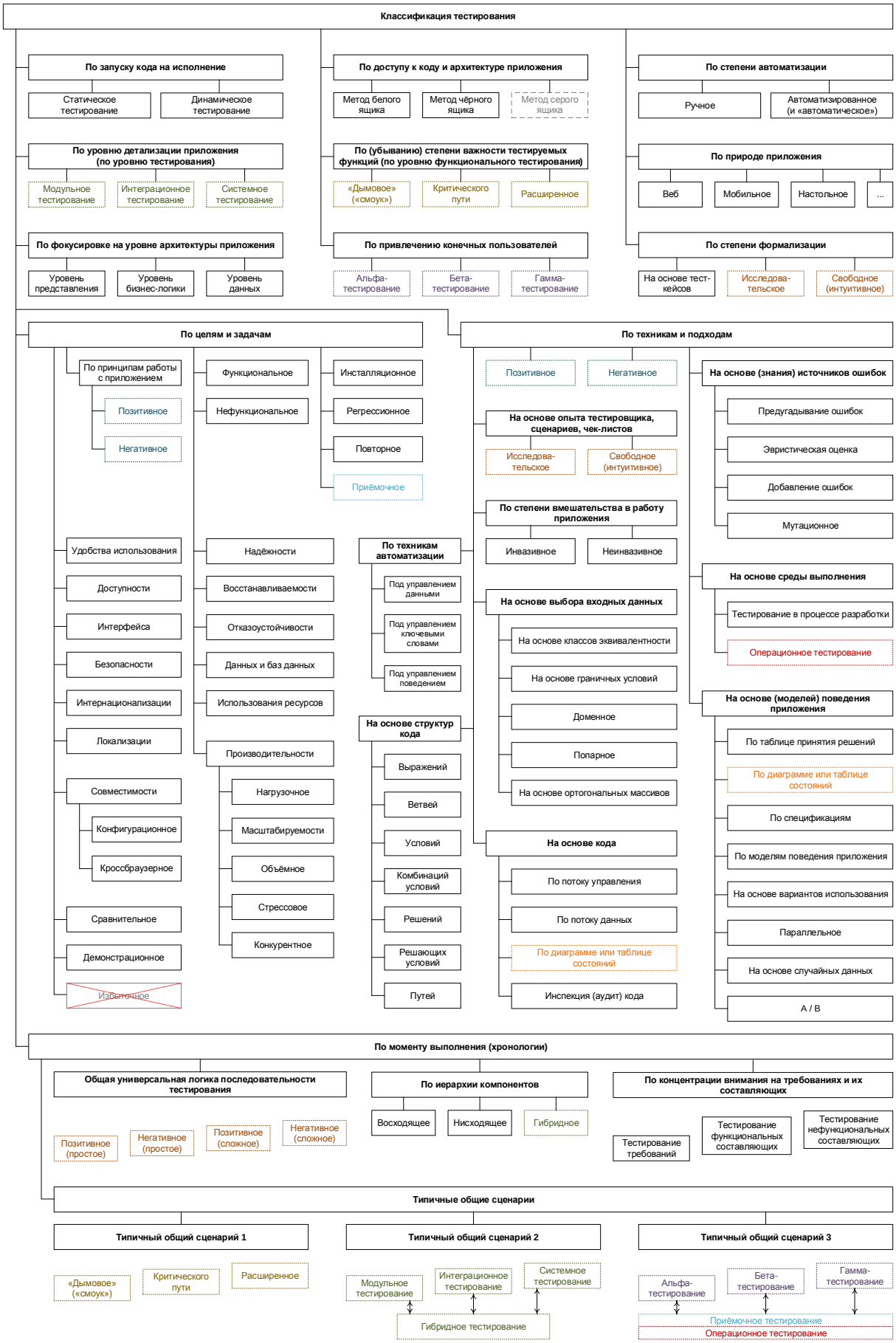


Рисунок 2.3.b — Подробная классификация тестирования (русскоязычный вариант)

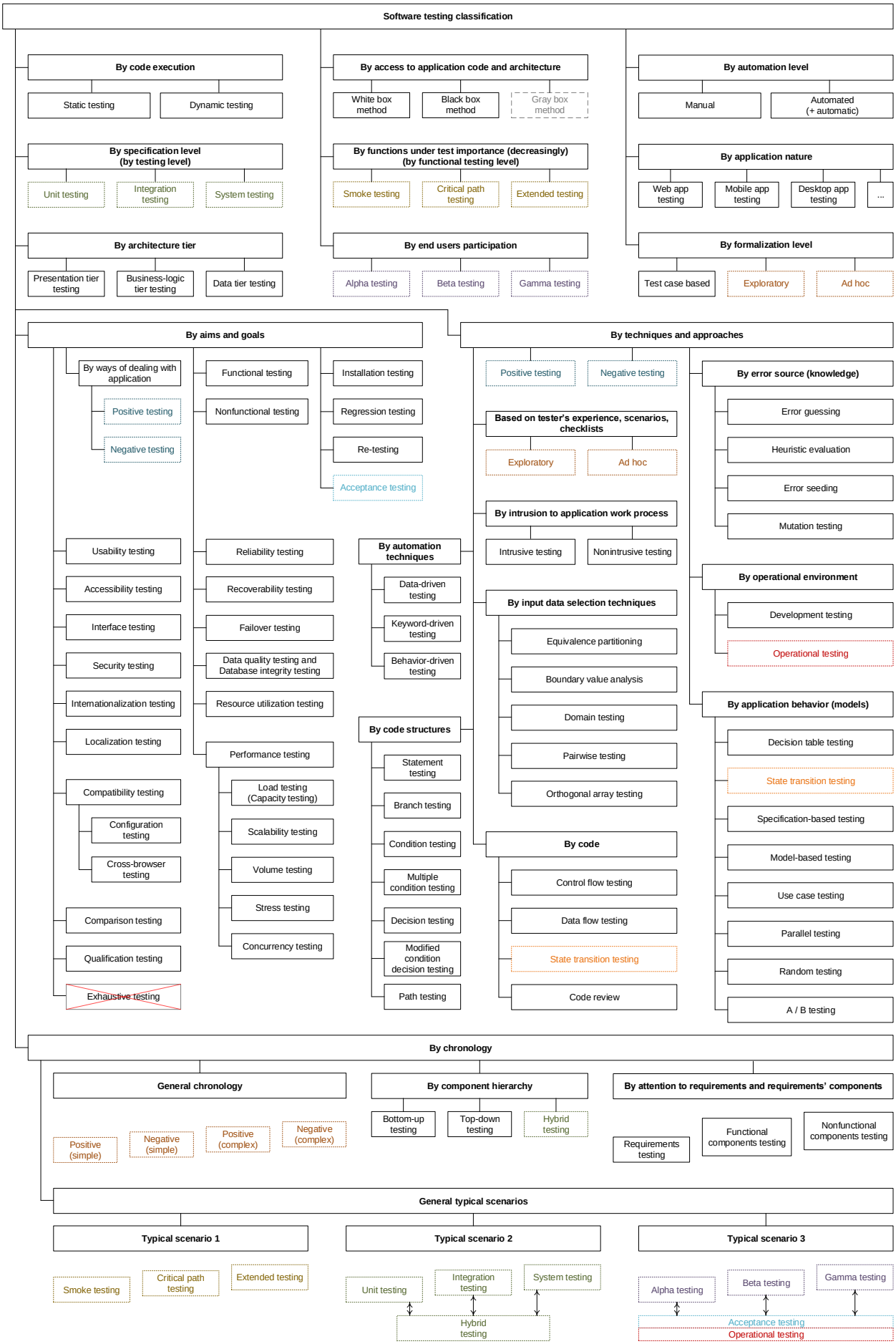


Рисунок 2.3.c — Подробная классификация тестирования (англоязычный вариант)

2.3.2.2. 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[112]) — 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[113]), являющегося специфической вариацией взаимного просмотра{45} в применении к исходному коду). Код приложения также можно проверять с использованием техник тестирования на основе структур кода{91}.
  - Параметры (настройки) среды исполнения приложения.
  - Подготовленные тестовые данные.
- **Динамическое тестирование** (dynamic testing[114]) — тестирование с запуском кода на исполнение. Запускаться на исполнение может как код всего приложения целиком (системное тестирование{71}), так и код нескольких взаимосвязанных частей (интеграционное тестирование{71}), отдельных частей (модульное или компонентное тестирование{71}) и даже отдельные участки кода. Основная идея этого вида тестирования состоит в том, что проверяется реальное поведение (части) приложения.

2.3.2.3. Классификация по доступу к коду и архитектуре приложения

- **Метод белого ящика** (white box testing[115], 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[116]). Для более глубокого изучения сути метода белого ящика рекомендуется ознакомиться с техниками исследования потока управления{90} или потока данных{90}, использования диаграмм состояний{90}. 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{71} и вовсе предполагает запуск кода на исполнение и при этом работу именно с кодом, а не с «приложением целиком»).

---

[112] **Static testing.** Testing of a software development artifact, e.g., requirements, design or code, without execution of these artifacts, e.g., reviews or static analysis. [ISTQB Glossary]

[113] Jason Cohen, «Best Kept Secrets of Peer Code Review (Modern Approach. Practical Advice.)». Официально распространяемую электронную версию книги можно взять здесь: http://smartbear.com/SmartBear/media/pdfs/best-kept-secrets-of-peer-code-review.pdf

[114] **Dynamic testing.** Testing that involves the execution of the software of a component or system. [ISTQB Glossary]

[115] **White box testing.** Testing based on an analysis of the internal structure of the component or system. [ISTQB Glossary]

[116] **Design-based Testing.** An approach to testing in which test cases are designed based on the architecture and/or detailed design of a component or system (e.g. tests of interfaces between components or systems). [ISTQB Glossary]

- **Метод чёрного ящика** (black box testing[117], closed box testing, specification-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. При этом абсолютное большинство перечисленных на рисунках 2.3.b и 2.3.c видов тестирования работают по методу чёрного ящика, идею которого в альтернативном определении можно сформулировать так: 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 чёрного ящика основной информацией для создания тест-кейсов выступает документация (особенно — требования (requirements-based testing[118])) 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[119]) — комбинация методов белого ящика и чёрного ящика, состоящая в том, что к части кода и архитектуры у тестировщика доступ есть, а к части — нет. На рисунках 2.3.b и 2.3.c этот метод обозначен особым пунктиром и серым цветом потому, что его явное упоминание — крайне редкий случай: обычно говорят о методах белого или чёрного ящика в применении к тем или иным частям приложения, при этом понимая, что «приложение целиком» тестируется по методу серого ящика.



Важно! Некоторые авторы[120] определяют метод серого ящика как противопоставление методам белого и чёрного ящика, особо подчёркивая, что при работе по методу серого ящика внутренняя структура тестируемого объекта известна частично и выясняется по мере исследования. Этот подход, бесспорно, имеет право на существование, но в своём предельном случае он вырождается до состояния «часть системы мы знаем, часть — не знаем», т.е. до всё той же комбинации белого и чёрного ящиков.

Если сравнить основные преимущества и недостатки перечисленных методов, получается следующая картина (см. таблицу 2.3.a).

Таблица 2.3.a — Преимущества и недостатки методов белого, чёрного и серого ящиков

	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.	• Не может выполняться тестировщиками, не обладающими достаточными знаниями в области программирования. • Тестирование сфокусировано на реализованной функциональности, что повышает вероятность

[117] **Black box testing.** Testing, either functional or non-functional, without reference to the internal structure of the component or system. [ISTQB Glossary]

[118] **Requirements-based Testing.** An approach to testing in which test cases are designed based on test objectives and test conditions derived from requirements, e.g. tests that exercise specific functions or probe non-functional attributes such as reliability or usability. [ISTQB Glossary]

[119] **Gray box testing** is a software testing method, which is a combination of Black Box Testing method and White Box Testing method. … In Gray Box Testing, the internal structure is partially known. This involves having access to internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level. [«Gray Box Testing Fundamentals», http://softwaretestingfundamentals.com/gray-box-testing].

[120] «Gray box testing (gray box) definition», Margaret Rouse [http://searchsoftwarequality.techtarget.com/definition/gray-box]

	• 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.	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{137}.
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{137}. • 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.	

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

2.3.2.4. Классификация по степени автоматизации

- **Ручное тестирование** (manual testing[121]) — 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[122]) — набор техник, подходов и инструментальных средств, позволяющий исключить человека из выполнения некоторых задач в процессе тестирования.

[121] **Manual testing** is performed by the tester who carries out all the actions on the tested application manually, step by step and indicates whether a particular step was accomplished successfully or whether it failed. Manual testing is always a part of any testing effort. It is especially useful in the initial phase of software development, when the software and its user interface are not stable enough, and beginning the automation does not make sense. (SmartBear TestComplete user manual, http://support.smartbear.com/viewarticle/55004/)

[122] **Test automation** is the use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions. Commonly, test automation involves automating a manual process already in place that uses a formalized testing process. (Ravinder Veer Hooda, “An Automation of Software Testing: A Foundation for the Future”)

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[109] 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 (см. таблицу 2.3.b).

Таблица 2.3.b — Преимущества и недостатки автоматизированного тестирования

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[123]), но при этом столь же ощутимо увеличивает риски.



Задание 2.3.a: сформируйте аналогичную таблицу с преимуществами и недостатками ручного тестирования. Подсказка: здесь недостаточно просто поменять заголовки колонок с преимуществами и недостатками автоматизации.

[123] **Coverage, Test coverage.** The degree, expressed as a percentage, to which a specified coverage item has been exercised by a test suite. [ISTQB Glossary]

2.3.2.5. 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[124]) 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{69}, значительно упрощающих и ускоряющих разработку соответствующих тест-кейсов.

  Из-за особенностей перевода на русский язык теряются нюансы степени детализации: «юнит-тестирование», как правило, направлено на тестирование атомарных участков кода, «модульное» — на тестирование классов и небольших библиотек, «компонентное» — на тестирование библиотек и структурных частей приложения. Но эта классификация не стандартизирована, и у различных авторов можно встретить совершенно разные взаимоисключающие трактовки.

- **Интеграционное тестирование** (integration testing[125], component integration testing[126], pairwise integration testing[127], system integration testing[128], incremental testing[129], interface testing[130], thread testing[131]) 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. (См. также техники восходящего, нисходящего и гибридного тестирования в хронологической классификации по иерархии компонентов{95}.)

---

[124] **Module testing, Unit testing, Component testing.** The testing of individual software components. [ISTQB Glossary]

[125] **Integration testing.** Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems. [ISTQB Glossary]

[126] **Component integration testing.** Testing performed to expose defects in the interfaces and interaction between integrated components. [ISTQB Glossary]

[127] **Pairwise integration testing.** A form of integration testing that targets pairs of components that work together, as shown in a call graph. [ISTQB Glossary]

[128] **System integration testing.** Testing the integration of systems and packages; testing interfaces to external organizations (e.g. Electronic Data Interchange, Internet). [ISTQB Glossary]

[129] **Incremental testing.** Testing where components or systems are integrated and tested one or some at a time, until all the components or systems are integrated and tested. [ISTQB Glossary]

[130] **Interface testing.** An integration test type that is concerned with testing the interfaces between components or systems. [ISTQB Glossary]

[131] **Thread testing.** An approach to component integration testing where the progressive integration of components follows the implementation of subsets of the requirements, as opposed to the integration of components by levels of a hierarchy. [ISTQB Glossary]

- **Системное тестирование** (system testing[132]) 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, перечисленных в данной главе.

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 2.3.d.

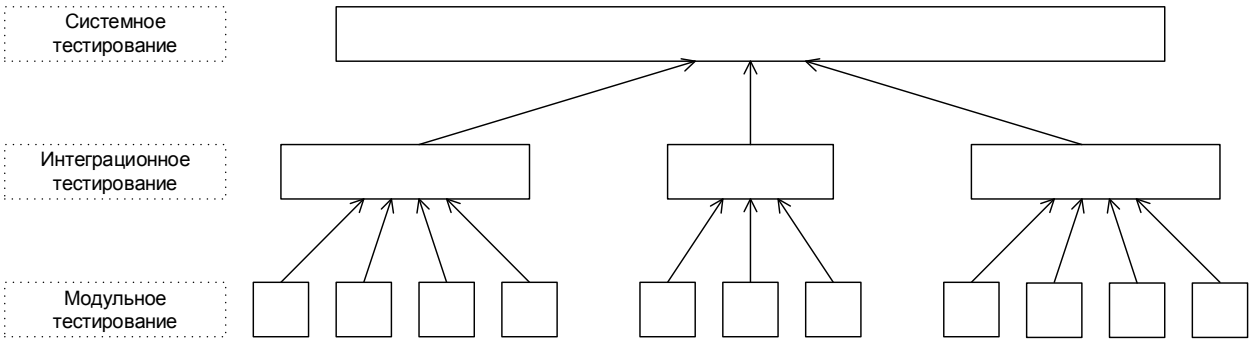


Рисунок 2.3.d — Схематичное представление классификации тестирования по уровню детализации приложения

Если обратиться к словарю ISTQB и прочитать определение уровня тестирования (test level[133]), то можно увидеть, что аналогичное разбиение на модульное, интеграционное и системное тестирование, к которым добавлено ещё и приёмочное тестирование{81}, используется в контексте разделения областей ответственности на проекте. Но такая классификация больше относится к вопросам управления проектом, чем к тестированию в чистом виде, а потому выходит за рамки рассматриваемых нами вопросов.



Самый полный вариант классификации тестирования по уровню тестирования можно посмотреть в статье «What are Software Testing Levels?[134]». Для улучшения запоминания отразим эту идею на рисунке 2.3.e, но отметим, что это скорее общий теоретический взгляд.

[132] **System testing.** The process of testing an integrated system to verify that it meets specified requirements. [ISTQB Glossary]

[133] **Test level.** A group of test activities that are organized and managed together. A test level is linked to the responsibilities in a project. Examples of test levels are component test, integration test, system test and acceptance test. [ISTQB Glossary]

[134] «What are Software Testing Levels?» [http://istqbexamcertification.com/what-are-software-testing-levels/]

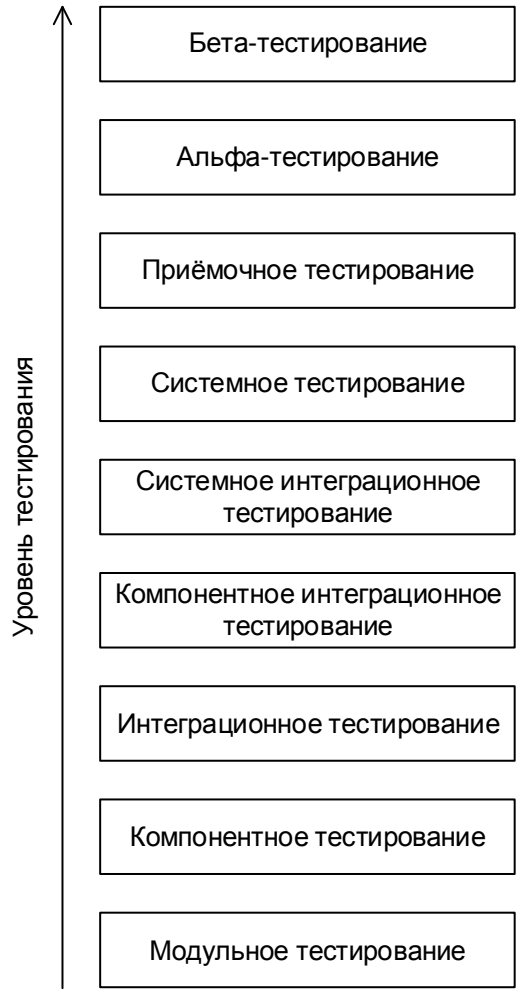


Рисунок 2.3.e — Самый полный вариант классификации тестирования по уровню тестирования

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

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

- **Дымовое тестирование** (smoke test[135], intake test[136], build verification test[137]) направлено на проверку самой главной, самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной

[135] **Smoke test, Confidence test, Sanity test.** A subset of all defined/planned test cases that cover the main functionality of a component or system, to ascertaining that the most crucial functions of a program work, but not bothering with finer details. [ISTQB Glossary]

[136] **Intake test.** A special instance of a smoke test to decide if the component or system is ready for detailed and further testing. An intake test is typically carried out at the start of the test execution phase. [ISTQB Glossary]

[137] **Build verification test.** A set of automated tests which validates the integrity of each new build and verifies its key/core functionality, stability and testability. It is an industry practice when a high frequency of build releases occurs (e.g., agile projects) and it is run on every new build before the build is released for further testing. [ISTQB Glossary]

саму идею использования приложения (или иного объекта, подвергаемого дымовому тестированию).



Внимание! Очень распространённая проблема! Из-за особенности перевода на русский язык под термином «приёмочное тестирование» часто может пониматься как «smoke test»{73}, так и «acceptance test»{81}, 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[138], что разница[139] есть и может быть выражена следующей схемой (рисунок 2.3.f):

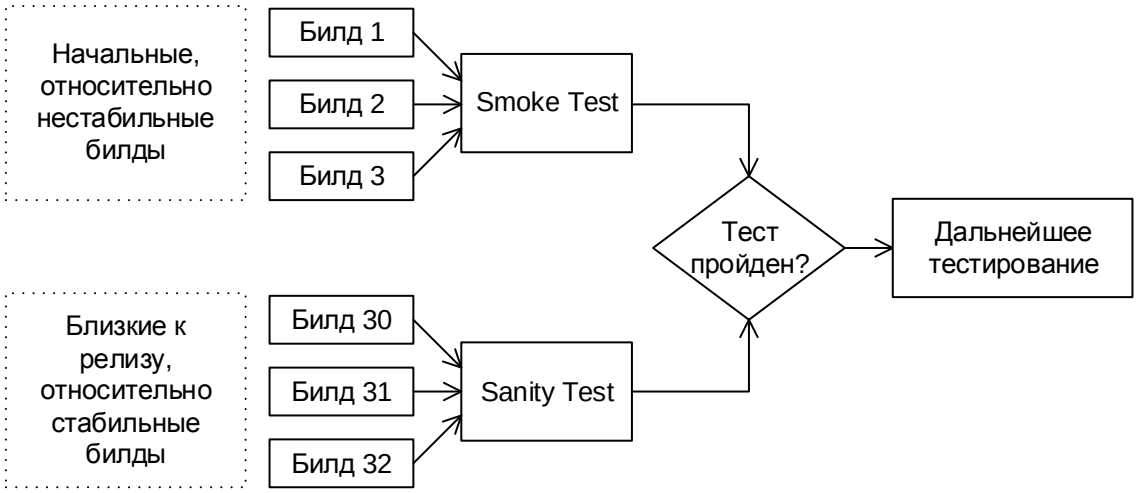


Рисунок 2.3.f — Трактовка разницы между smoke test и sanity test

- **Тестирование критического пути** (critical path[140] 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

[138] «Smoke Vs Sanity Testing — Introduction and Differences» [http://www.guru99.com/smoke-sanity-testing.html]

[139] «Smoke testing and sanity testing — Quick and simple differences» [http://www.softwaretestinghelp.com/smoke-testing-and-sanity-testing-difference/]

[140] **Critical path.** Longest sequence of activities in a project plan which must be completed on time for the project to complete on due date. An activity on the critical path cannot be started until its predecessor activity is complete; if it is delayed for a day, the entire project will be delayed for a day unless the activity following the delayed activity is completed a day earlier. [http://www.business-dictionary.com/definition/critical-path.html]

подмножество функций приложения (см. рисунок 2.3.g). 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).

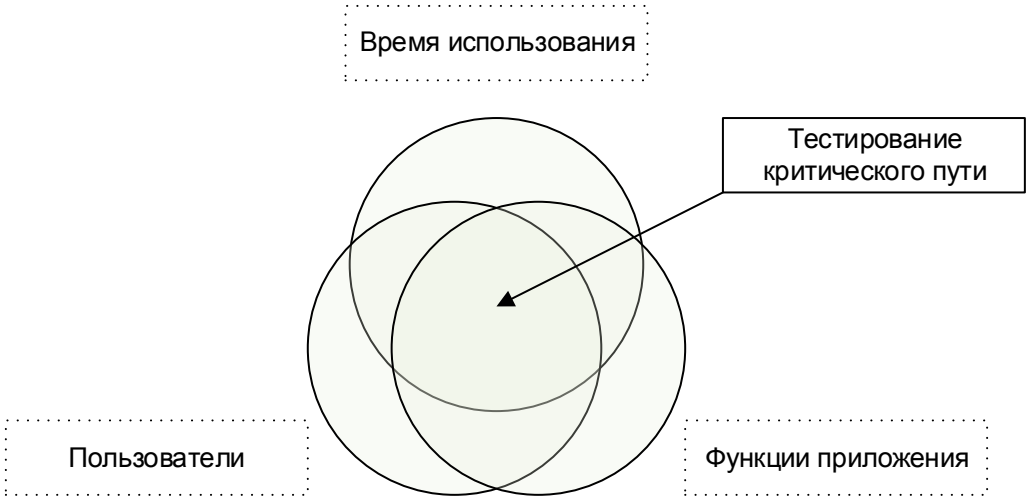


Рисунок 2.3.g — Суть тестирования критического пути

- **Расширенное тестирование** (extended test[141]) 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. 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).



К сожалению, часто можно встретить мнение, что дымовое тестирование, тестирование критического пути и расширенное тестирование напрямую связаны с позитивным{76} тестированием и негативным{76} 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.

[141] **Extended test.** The idea is to develop a comprehensive application system test suite by modeling essential capabilities as extended use cases. [By «Extended Use Case Test Design Pattern», Rob Kuijt]

Для лучшего запоминания отразим эту классификацию графически:

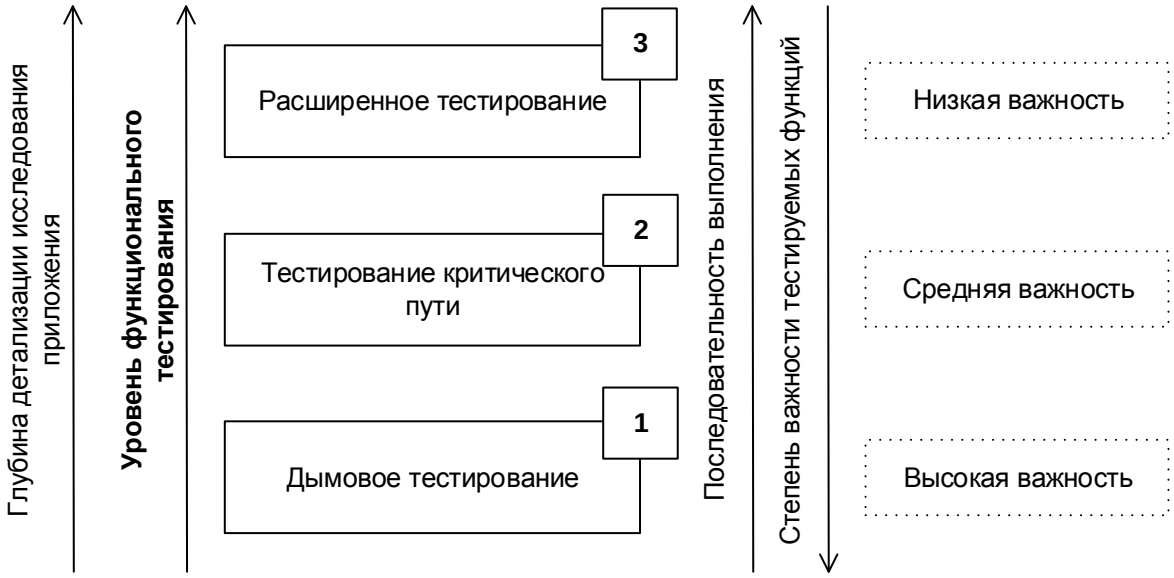


Рисунок 2.3.h — Классификация тестирования по (убыванию) степени важности тестируемых функций (по уровню функционального тестирования)

2.3.2.7. Классификация по принципам работы с приложением

- **Позитивное тестирование** (positive testing[142]) 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[143], invalid testing[144]) — 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.

[142] **Positive testing** is testing process where the system validated against the valid input data. In this testing tester always check for only valid set of values and check if application behaves as expected with its expected inputs. The main intention of this testing is to check whether software application not showing error when not supposed to & showing error when supposed to. Such testing is to be carried out keeping positive point of view & only execute the positive scenario. Positive Testing always tries to prove that a given product and project always meets the requirements and specifications. [http://www.softwaretestingclass.com/positive-and-negative-testing-in-software-testing/]

[143] **Negative testing.** Tests aimed at showing that a component or system does not work. Negative testing is related to the testers' attitude rather than a specific test approach or test design technique, e.g. testing with invalid input values or exceptions. [ISTQB Glossary]

[144] **Invalid testing.** Testing using input values that should be rejected by the component or system. [ISTQB Glossary]

2.3.2.8. 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{83} (в особенности — кросс-браузерного тестирования{83}), тестирования производительности{85}, автоматизации тестирования с использованием широкого спектра инструментальных средств.
- **Тестирование мобильных приложений** (mobile applications testing) также требует повышенного внимания к тестированию совместимости{83}, оптимизации производительности{85} (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.

2.3.2.9. 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, ознакомьтесь с ним хотя бы по материалу[145] из Википедии.

[145] «Multitier architecture», Wikipedia [http://en.wikipedia.org/wiki/Multitier_architecture]

2.3.2.10. Классификация по привлечению конечных пользователей

Все три перечисленных ниже вида тестирования относятся к операционному тестированию{81}.

- **Альфа-тестирование** (alpha testing[146]) выполняется внутри организации-разработчика с возможным частичным привлечением конечных пользователей. Может являться формой внутреннего приёмочного тестирования{81}. 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[147]) выполняется вне организации-разработчика с активным привлечением конечных пользователей/заказчиков. Может являться формой внешнего приёмочного тестирования{81}. 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[148]) — 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{81}. Суть этого вида вкратце: продукт уже почти готов, и сейчас обратная связь от реальных пользователей используется для устранения последних недоработок.

2.3.2.11. Классификация по степени формализации

- **Тестирование на основе тест-кейсов** (scripted testing[149], 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[150]) — частично формализованный подход, в рамках которого тестировщик выполняет работу с

[146] **Alpha testing.** Simulated or actual operational testing by potential users/customers or an independent test team at the developers' site, but outside the development organization. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing. [ISTQB Glossary]

[147] **Beta testing.** Operational testing by potential and/or existing users/customers at an external site not otherwise involved with the developers, to determine whether or not a component or system satisfies the user/customer needs and fits within the business processes. Beta testing is often employed as a form of external acceptance testing for off-the-shelf software in order to acquire feedback from the market. [ISTQB Glossary]

[148] **Gamma testing** is done when software is ready for release with specified requirements, this testing done directly by skipping all the in-house testing activities. The software is almost ready for final release. No feature development or enhancement of the software is undertaken and tightly scoped bug fixes are the only code. Gamma check is performed when the application is ready for release to the specified requirements and this check is performed directly without going through all the testing activities at home. [http://www.360logica.com/blog/2012/06/what-are-alpha-beta-and-gamma-testing.html]

[149] **Scripted testing.** Test execution carried out by following a previously documented sequence of tests. [ISTQB Glossary]

[150] **Exploratory testing.** An informal test design technique where the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests. [ISTQB Glossary]

приложением по выбранному сценарию{137}, 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[151]). 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[152]).



Дополнительную информацию об исследовательском тестировании можно получить из статьи Джеймса Баха «Что такое исследовательское тестирование?[153]»

- **Свободное (интуитивное) тестирование** (ad hoc testing[154]) — 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[155]) 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.

2.3.2.12. Классификация по целям и задачам

- **Позитивное тестирование** (рассмотрено ранее{76}).
- **Негативное тестирование** (рассмотрено ранее{76}).
- **Функциональное тестирование** (functional testing[156]) — вид тестирования, направленный на проверку корректности работы функциональности приложения (корректность реализации функциональных требований{37}). Часто функциональное тестирование ассоциируют с тестированием по методу чёрного ящика{68}, однако и по методу белого ящика{67} вполне можно проверять корректность реализации функциональности.

---

151 **Session-based Testing.** An approach to testing in which test activities are planned as uninterrupted sessions of test design and execution, often used in conjunction with exploratory testing. [ISTQB Glossary]

152 **Checklist-based Testing.** An experience-based test design technique whereby the experienced tester uses a high-level list of items to be noted, checked, or remembered, or a set of rules or criteria against which a product has to be verified. [ISTQB Glossary]

153 «What is Exploratory Testing?», James Bach [http://www.satisfice.com/articles/what_is_et.shtml]

154 **Ad hoc testing.** Testing carried out informally; no formal test preparation takes place, no recognized test design technique is used, there are no expectations for results and arbitrariness guides the test execution activity. [ISTQB Glossary]

155 **Experience-based Testing.** Testing based on the tester's experience, knowledge and intuition. [ISTQB Glossary]

156 **Functional testing.** Testing based on an analysis of the specification of the functionality of a component or system. [ISTQB Glossary]



Часто возникает вопрос, в чём разница между функциональным тестированием (functional testing[156]) и тестированием функциональности (functionality testing[157]). Подробнее о функциональном тестировании можно прочесть в статье «What is Functional testing (Testing of functions) in software?»[158], а о тестировании функциональности в статье «What is functionality testing in software?»[159].
Если вкратце, то:

- функциональное тестирование (как антоним нефункционального) направлено на проверку того, какие функции приложения реализованы, и что они работают верным образом;
- тестирование функциональности направлено на те же задачи, но акцент смещён в сторону исследования приложения в реальной рабочей среде, после локализации и в тому подобных ситуациях.

- **Нефункциональное тестирование** (non-functional testing[160]) — вид тестирования, направленный на проверку нефункциональных особенностей приложения (корректность реализации нефункциональных требований{37}), таких как удобство использования, совместимость, производительность, безопасность и т.д.

- **Инсталляционное тестирование** (installation testing, installability testing[161]) — 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:
  - новая среда исполнения, в которой приложение ранее не было инсталлировано;
  - обновление существующей версии («апгрейд»);
  - изменение текущей версии на более старую («даунгрейд»);
  - повторная установка приложения с целью устранения возникших проблем («переинсталляция»);
  - повторный запуск инсталляции после ошибки, приведшей к невозможности продолжения инсталляции;
  - удаление приложения;
  - установка нового приложения из семейства приложений;
  - автоматическая инсталляция без участия пользователя.

- **Регрессионное тестирование** (regression testing[162]) — тестирование, направленное на проверку того факта, что в ранее работоспособной функциональности не появились ошибки, вызванные изменениями в приложении

[157] **Functionality testing.** The process of testing to determine the functionality of a software product (the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions). [ISTQB Glossary]

[158] «What is Functional testing (Testing of functions) in software?» [http://istqbexamcertification.com/what-is-functional-testing-testing-of-functions-in-software/]

[159] «What is functionality testing in software?» [http://istqbexamcertification.com/what-is-functionality-testing-in-software/]

[160] **Non-functional testing.** Testing the attributes of a component or system that do not relate to functionality, e.g. reliability, efficiency, usability, maintainability and portability. [ISTQB Glossary]

[161] **Installability testing.** The process of testing the installability of a software product. Installability is the capability of the software product to be installed in a specified environment. [ISTQB Glossary]

[162] **Regression testing.** Testing of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made. It is performed when the software or its environment is changed. [ISTQB Glossary]

или среде его функционирования. Фредерик Брукс в своей книге «Мифический человеко-месяц»[163] 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[164], 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{160}, направленным на то, чтобы перевести дефект в состояние «проверен» и «закрыт».
- **Приёмочное тестирование** (acceptance testing[165]) — 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»):
  - **Производственное приёмочное тестирование** (factory acceptance testing[166]) — 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{78}.
  - **Операционное приёмочное тестирование** (operational acceptance testing[167], production acceptance testing) — операционное тестирование{81}, выполняемое с точки зрения выполнения инсталляции, потребления приложением ресурсов, совместимости с программной и аппаратной платформой и т.д.
  - **Итоговое приёмочное тестирование** (site acceptance testing[168]) — 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[169]) — тестирование, проводимое в реальной или приближенной к реальной операционной среде (operational environment[170]), включающей операционную систему, системы

[163] Frederick Brooks, «The Mythical Man-Month».

[164] **Re-testing, Confirmation testing.** Testing that runs test cases that failed the last time they were run, in order to verify the success of corrective actions. [ISTQB Glossary]

[165] **Acceptance Testing.** Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system. [ISTQB Glossary]

[166] **Factory acceptance testing.** Acceptance testing conducted at the site at which the product is developed and performed by employees of the supplier organization, to determine whether or not a component or system satisfies the requirements, normally including hardware as well as software. [ISTQB Glossary]

[167] **Operational acceptance testing, Production acceptance testing.** Operational testing in the acceptance test phase, typically performed in a (simulated) operational environment by operations and/or systems administration staff focusing on operational aspects, e.g. recoverability, resource-behavior, installability and technical compliance. [ISTQB Glossary]

[168] **Site acceptance testing.** Acceptance testing by users/customers at their site, to determine whether or not a component or system satisfies the user/customer needs and fits within the business processes, normally including hardware as well as software. [ISTQB Glossary]

[169] **Operational testing.** Testing conducted to evaluate a component or system in its operational environment. [ISTQB Glossary]

[170] **Operational environment.** Hardware and software products installed at users' or customers' sites where the component or system under test will be used. The software may include operating systems, database management systems, and other applications. [ISTQB Glossary]

управления базами данных, серверы приложений, веб-серверы, аппаратное обеспечение и т.д.

- **Тестирование удобства использования** (usability[171] testing) — тестирование, направленное на исследование того, насколько конечному пользователю понятно, как работать с продуктом (understandability[172], learnability[173], operability[174]), а также на то, насколько ему нравится использовать продукт (attractiveness[175]). 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.



Важно! Тестирование удобства использования (usability[171] testing) и тестирование интерфейса пользователя (GUI testing[180]) — не одно и то же! Например, корректно работающий интерфейс может быть неудобным, а удобный может работать некорректно.

- **Тестирование доступности** (accessibility testing[176]) — тестирование, направленное на исследование пригодности продукта к использованию людьми с ограниченными возможностями (слабым зрением и т.д.)

- **Тестирование интерфейса** (interface testing[177]) — тестирование, направленное на проверку интерфейсов приложения или его компонентов. По определению ISTQB-глоссария этот вид тестирования относится к интеграционному тестированию{71}, и это вполне справедливо для таких его вариаций как тестирование интерфейса прикладного программирования (API testing[178]) и интерфейса командной строки (CLI testing[179]), 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[180]).



Важно! Тестирование интерфейса пользователя (GUI testing[180]) и тестирование удобства использования (usability[171] testing) — не одно и то же! Например, удобный интерфейс может работать некорректно, а корректно работающий интерфейс может быть неудобным.

---

171 **Usability.** The capability of the software to be understood, learned, used and attractive to the user when used under specified conditions. [ISTQB Glossary]

172 **Understandability.** The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use. [ISTQB Glossary]

173 **Learnability.** The capability of the software product to enable the user to learn its application. [ISTQB Glossary]

174 **Operability.** The capability of the software product to enable the user to operate and control it. [ISTQB Glossary]

175 **Attractiveness.** The capability of the software product to be attractive to the user. [ISTQB Glossary]

176 **Accessibility testing.** Testing to determine the ease by which users with disabilities can use a component or system. [ISTQB Glossary]

177 **Interface Testing.** An integration test type that is concerned with testing the interfaces between components or systems. [ISTQB Glossary]

178 **API testing.** Testing performed by submitting commands to the software under test using programming interfaces of the application directly. [ISTQB Glossary]

179 **CLI testing.** Testing performed by submitting commands to the software under test using a dedicated command-line interface. [ISTQB Glossary]

180 **GUI testing.** Testing performed by interacting with the software under test via the graphical user interface. [ISTQB Glossary]

- **Тестирование безопасности** (security testing[181]) — тестирование, направленное на проверку способности приложения противостоять злонамеренным попыткам получения доступа к данным или функциям, права на доступ к которым у злоумышленника нет.



Подробнее про этот вид тестирования можно почитать в статье «What is Security testing in software testing?»[182].

- **Тестирование интернационализации** (internationalization testing, i18n testing, globalization[183] testing, localizability[184] 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, см. следующий пункт), 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 и т.д.)

- **Тестирование локализации** (localization testing[185], l10n) — 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 (см. предыдущий пункт) и проверяет корректность перевода и адаптации продукта, а не готовность продукта к таким действиям.

- **Тестирование совместимости** (compatibility testing, interoperability testing[186]) — тестирование, направленное на проверку способности приложения работать в указанном окружении. Здесь может проверяться:
  - Совместимость с аппаратной платформой, операционной системой и сетевой инфраструктурой (конфигурационное тестирование, configuration testing[187]).
  - Совместимость с браузерами и их версиями (кросс-браузерное тестирование, cross-browser testing[188]). (См. также тестирование веб-приложений{77}).

[181] **Security testing.** Testing to determine the security of the software product. [ISTQB Glossary]

[182] «What is Security testing in software testing?» [http://istqbexamcertification.com/what-is-security-testing-in-software/]

[183] **Globalization.** The process of developing a program core whose features and code design are not solely based on a single language or locale. Instead, their design is developed for the input, display, and output of a defined set of Unicode-supported language scripts and data related to specific locales. [«Globalization Step-by-Step», https://msdn.microsoft.com/en-us/goglobal/bb688112]

[184] **Localizability.** The design of the software code base and resources such that a program can be localized into different language editions without any changes to the source code. [«Globalization Step-by-Step», https://msdn.microsoft.com/en-us/goglobal/bb688112]

[185] **Localization testing** checks the quality of a product's localization for a particular target culture/locale. This test is based on the results of globalization testing, which verifies the functional support for that particular culture/locale. Localization testing can be executed only on the localized version of a product. [«Localization Testing», https://msdn.microsoft.com/en-us/library/aa292138%28v=vs.71%29.aspx]

[186] **Compatibility Testing, Interoperability Testing.** The process of testing to determine the interoperability of a software product (the capability to interact with one or more specified components or systems). [ISTQB Glossary]

[187] **Configuration Testing, Portability Testing.** The process of testing to determine the portability of a software product (the ease with which the software product can be transferred from one hardware or software environment to another). [ISTQB Glossary]

[188] **Cross-browser testing** helps you ensure that your web site or web application functions correctly in various web browsers. Typically, QA engineers create individual tests for each browser or create tests that use lots of conditional statements that check the browser type used and execute browser-specific commands. [http://support.smartbear.com/viewarticle/55299/]

- Совместимость с мобильными устройствами (mobile testing[189]). (См. также тестирование мобильных приложений{77}).
- И так далее.

В некоторых источниках к тестированию совместимости добавляют (хоть и подчёркивая, что это — не его часть) т.н. тестирование соответствия (compliance testing[190], conformance testing, regulation testing).



Рекомендуется ознакомиться с дополнительным материалом по тестированию совместимости с мобильными платформами в статьях «What is Mobile Testing?»[191] и «Beginner's Guide to Mobile Application Testing»[192].

- **Тестирование данных** (data quality[193] testing) и баз данных (database integrity testing[194]) — 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[195], efficiency testing[196], storage testing[197]) — 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{85}.
- **Сравнительное тестирование** (comparison testing[198]) — тестирование, направленное на сравнительный анализ преимуществ и недостатков разрабатываемого продукта по отношению к его основным конкурентам.

---

189 **Mobile testing** is a testing with multiple operating systems (and different versions of each OS, especially with Android), multiple devices (different makes and models of phones, tablets, phablets), multiple carriers (including international ones), multiple speeds of data transference (3G, LTE, Wi-Fi), multiple screen sizes (and resolutions and aspect ratios), multiple input controls (including BlackBerry's eternal physical keypads), and multiple technologies — GPS, accelerometers — that web and desktop apps almost never use. [http://smartbear.com/all-resources/articles/what-is-mobile-testing/]

190 **Compliance testing, Conformance testing, Regulation testing.** The process of testing to determine the compliance of the component or system (the capability to adhere to standards, conventions or regulations in laws and similar prescriptions). [ISTQB Glossary]

191 «What is Mobile Testing?» [http://smartbear.com/all-resources/articles/what-is-mobile-testing/]

192 «Beginner's Guide to Mobile Application Testing» [http://www.softwaretestinghelp.com/beginners-guide-to-mobile-application-testing/]

193 **Data quality.** An attribute of data that indicates correctness with respect to some pre-defined criteria, e.g., business expectations, requirements on data integrity, data consistency. [ISTQB Glossary]

194 **Database integrity testing.** Testing the methods and processes used to access and manage the data(base), to ensure access methods, processes and data rules function as expected and that during access to the database, data is not corrupted or unexpectedly deleted, updated or created. [ISTQB Glossary]

195 **Resource utilization testing, Storage testing.** The process of testing to determine the resource-utilization of a software product. [ISTQB Glossary]

196 **Efficiency testing.** The process of testing to determine the efficiency of a software product (the capability of a process to produce the intended outcome, relative to the amount of resources used). [ISTQB Glossary]

197 **Storage testing.** This is a determination of whether or not certain processing conditions use more storage (memory) than estimated. [«Software Testing Concepts And Tools», Nageshwar Rao Pusuluri]

198 **Comparison testing.** Testing that compares software weaknesses and strengths to those of competitors' products. [«Software Testing and Quality Assurance», Jyoti J. Malhotra, Bhavana S. Tiple]

- **Демонстрационное тестирование** (qualification testing[199]) — формальный процесс демонстрации заказчику продукта с целью подтверждения, что продукт соответствует всем заявленным требованиям. В отличие от приёмочного тестирования{81} этот процесс более строгий и всеобъемлющий, но может проводиться и на промежуточных стадиях разработки продукта.
- **Избыточное тестирование** (exhaustive testing[200]) — 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[201]) — тестирование способности приложения выполнять свои функции в заданных условиях на протяжении заданного времени или заданного количества операций.
- **Тестирование восстанавливаемости** (recoverability testing[202]) — 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[203]) — 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[204]) — исследование показателей скорости реакции приложения на внешние воздействия при различной по характеру и интенсивности нагрузке. В рамках тестирования производительности выделяют следующие подвиды:
  - **Нагрузочное тестирование** (load testing[205], capacity testing[206]) — исследование способности приложения сохранять заданные показатели качества при нагрузке в допустимых пределах и некотором превышении этих пределов (определение «запаса прочности»).

[199] **Qualification testing.** Formal testing, usually conducted by the developer for the consumer, to demonstrate that the software meets its specified requirements. [«Software Testing Concepts And Tools», Nageshwar Rao Pusuluri]

[200] **Exhaustive testing.** A test approach in which the test suite comprises all combinations of input values and preconditions. [ISTQB Glossary]

[201] **Reliability Testing.** The process of testing to determine the reliability of a software product (the ability of the software product to perform its required functions under stated conditions for a specified period of time, or for a specified number of operations). [ISTQB Glossary]

[202] **Recoverability Testing.** The process of testing to determine the recoverability of a software product (the capability of the software product to re-establish a specified level of performance and recover the data directly affected in case of failure). [ISTQB Glossary]

[203] **Failover Testing.** Testing by simulating failure modes or actually causing failures in a controlled environment. Following a failure, the failover mechanism is tested to ensure that data is not lost or corrupted and that any agreed service levels are maintained (e.g., function availability or response times). [ISTQB Glossary]

[204] **Performance Testing.** The process of testing to determine the performance of a software product. [ISTQB Glossary]

[205] **Load Testing.** A type of performance testing conducted to evaluate the behavior of a component or system with increasing load, e.g. numbers of parallel users and/or numbers of transactions, to determine what load can be handled by the component or system. [ISTQB Glossary]

[206] **Capacity Testing.** Testing to determine how many users and/or transactions a given system will support and still meet performance goals. [https://msdn.microsoft.com/en-us/library/bb924357.aspx]

- **Тестирование масштабируемости** (scalability testing[207]) — исследование способности приложения увеличивать показатели производительности в соответствии с увеличением количества доступных приложению ресурсов.
- **Объёмное тестирование** (volume testing[208]) — исследование производительности приложения при обработке различных (как правило, больших) объёмов данных.
- **Стрессовое тестирование** (stress testing[209]) — 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[210]) и представляет собой крайнюю форму негативного тестирования{76}.
- **Конкурентное тестирование** (concurrency testing[211]) — 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.

В качестве отдельных или вспомогательных техник в рамках тестирования производительности могут использоваться тестирование использования ресурсов{84}, тестирование надёжности{85}, тестирование восстанавливаемости{85}, тестирование отказоустойчивости{85} и т.д.



Подробное рассмотрение нескольких видов тестирования производительности приведено в статьях «Автоматизация тестирования производительности: основные положения и области применения» [212] и «Description of typical performance test program»[213].

[207] **Scalability Testing.** Testing to determine the scalability of the software product (the capability of the software product to be upgraded to accommodate increased loads). [ISTQB Glossary]

[208] **Volume Testing.** Testing where the system is subjected to large volumes of data. [ISTQB Glossary]

[209] **Stress testing.** A type of performance testing conducted to evaluate a system or component at or beyond the limits of its anticipated or specified workloads, or with reduced availability of resources such as access to memory or servers. [ISTQB Glossary]

[210] **Destructive software testing** assures proper or predictable software behavior when the software is subject to improper usage or improper input, attempts to crash a software product, tries to crack or break a software product, checks the robustness of a software product. [«Towards Destructive Software Testing», Kiumi Akingbehin]

[211] **Concurrency testing**. Testing to determine how the occurrence of two or more activities within the same interval of time, achieved either by interleaving the activities or by simultaneous execution, is handled by the component or system. [ISTQB Glossary]

[212] «Автоматизация тестирования производительности: основные положения и области применения» [http://svyatoslav.biz/technologies/performance_testing/]

[213] «Description of typical performance test program» [http://benchmark.reutersinsider.com/Description_of_typical_performance_test_program.html]

2.3.2.13. Классификация по техникам и подходам

- **Позитивное тестирование** (рассмотрено ранее{76}).
- **Негативное тестирование** (рассмотрено ранее{76}).
- Тестирование на основе опыта тестировщика, сценариев, чек-листов:
  - **Исследовательское тестирование** (рассмотрено ранее{78}).
  - **Свободное (интуитивное) тестирование** (рассмотрено ранее{79}).
- Классификация по степени вмешательства в работу приложения:
  - **Инвазивное тестирование** (intrusive testing[214]) — 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[215]) в сам код приложения (например, для анализа работы приложения было добавлено дополнительное протоколирование, включён вывод отладочной информации и т.д.) Некоторые источники рассматривают[216] инвазивное тестирование как форму негативного{76} или даже стрессового{86} тестирования.
  - **Неинвазивное тестирование** (nonintrusive testing[217]) — тестирование, выполнение которого незаметно для приложения и не влияет на процесс его обычной работы.
- Классификация по техникам автоматизации:
  - **Тестирование под управлением данными** (data-driven testing[218]) — 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[219]) — 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 (командами).
  - **Тестирование под управлением поведением** (behavior driven testing[220]) — способ разработки автоматизированных тест-кейсов, в котором основное внимание уделяется корректности работы бизнес-сценариев, а не отдельным деталям функционирования приложения.

---

[214] **Intrusive testing.** Testing that collects timing and processing information during program execution that may change the behavior of the software from its behavior in a real environment. Intrusive testing usually involves additional code embedded in the software being tested or additional processes running concurrently with software being tested on the same processor. [http://encyclopedia2.thefreedictionary.com/intrusive+testing]

[215] **Level of intrusion.** The level to which a test object is modified by adjusting it for testability. [ISTQB Glossary]

[216] Intrusive testing can be considered a type of interrupt testing, which is used to test how well a system reacts to intrusions and interrupts to its normal workflow. [http://www.techopedia.com/definition/7802/intrusive-testing]

[217] **Nonintrusive Testing.** Testing that is transparent to the software under test, i.e., does not change its timing or processing characteristics. Nonintrusive testing usually involves additional hardware that collects timing or processing information and processes that information on another platform. [http://encyclopedia2.thefreedictionary.com/nonintrusive+testing]

[218] **Data-driven Testing (DDT).** A scripting technique that stores test input and expected results in a table or spreadsheet, so that a single control script can execute all of the tests in the table. Data-driven testing is often used to support the application of test execution tools such as capture/playback tools. [ISTQB Glossary]

[219] **Keyword-driven Testing (KDT).** A scripting technique that uses data files to contain not only test data and expected results, but also keywords related to the application being tested. The keywords are interpreted by special supporting scripts that are called by the control script or the test. [ISTQB Glossary]

[220] **Behavior-driven Testing (BDT).** Behavior Driven Tests focuses on the behavior rather than the technical implementation of the software. If you want to emphasize on business point of view and requirements then BDT is the way to go. BDT are Given-when-then style tests written in natural language which are easily understandable to non-technical individuals. Hence these tests allow business analysts and management people to actively participate in test creation and review process. [Jyothi Rangaiah, http://www.womentesters.com/behaviour-driven-testing-an-introduction/]

- Классификация на основе (знания) источников ошибок:
  - **Тестирование предугадыванием ошибок** (error guessing[221]) — 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[222]), в котором новые тесты строятся на основе информации о ранее обнаруженных в приложении проблемах.
  - **Эвристическая оценка** (heuristic evaluation[223]) — техника тестирования удобства использования{82}, направленная на поиск проблем в интерфейсе пользователя, представляющих собой отклонение от общепринятых норм.
  - **Мутационное тестирование** (mutation testing[224]) — 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).
  - **Тестирование добавлением ошибок** (error seeding[225]) — 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:
  - **Тестирование на основе классов эквивалентности** (equivalence partitioning[226]) — 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.

[221] **Error Guessing.** A test design technique where the experience of the tester is used to anticipate what defects might be present in the component or system under test as a result of errors made, and to design tests specifically to expose them. [ISTQB Glossary]

[222] **Failure-directed Testing.** Software testing based on the knowledge of the types of errors made in the past that are likely for the system under test. [http://dictionary.reference.com/browse/failure-directed+testing].

[223] **Heuristic Evaluation.** A usability review technique that targets usability problems in the user interface or user interface design. With this technique, the reviewers examine the interface and judge its compliance with recognized usability principles (the «heuristics»). [ISTQB Glossary]

[224] **Mutation Testing, Back-to-Back Testing.** Testing in which two or more variants of a component or system are executed with the same inputs, the outputs compared, and analyzed in cases of discrepancies. [ISTQB Glossary]

[225] **Error seeding.** The process of intentionally adding known faults to those already in a computer program for the purpose of monitoring the rate of detection and removal, and estimating the number of faults remaining in the program. [ISTQB Glossary]

[226] **Equivalence partitioning.** A black box test design technique in which test cases are designed to execute representatives from equivalence partitions. In principle test cases are designed to cover each partition at least once. [ISTQB Glossary]

- **Тестирование на основе граничных условий** (boundary value analysis[227]) — 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.
- **Доменное тестирование** (domain analysis[228], 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.
- **Попарное тестирование** (pairwise testing[229]) — 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[230]) 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).



Попарное тестирование (pairwise testing[229]) — это НЕ парное тестирование (pair testing[231])!

- **Тестирование на основе ортогональных массивов** (orthogonal array testing[232]) — инструментальная техника попарного и N-комбинационного 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).



Ортогональные массивы — это НЕ ортогональные матрицы. Это совершенно разные понятия! Сравните их описания в статьях «Orthogonal array»[233] и «Orthogonal matrix»[234].

[227] **Boundary value analysis.** A black box test design technique in which test cases are designed based on boundary values (input values or output values which are on the edge of an equivalence partition or at the smallest incremental distance on either side of an edge, for example the minimum or maximum value of a range). [ISTQB Glossary]

[228] **Domain analysis.** A black box test design technique that is used to identify efficient and effective test cases when multiple variables can or should be tested together. It builds on and generalizes equivalence partitioning and boundary values analysis. [ISTQB Glossary]

[229] **Pairwise testing.** A black box test design technique in which test cases are designed to execute all possible discrete combinations of each pair of input parameters. [ISTQB Glossary]

[230] **N-wise testing.** A black box test design technique in which test cases are designed to execute all possible discrete combinations of any set of n input parameters. [ISTQB Glossary]

[231] **Pair testing.** Two persons, e.g. two testers, a developer and a tester, or an end-user and a tester, working together to find defects. Typically, they share one computer and trade control of it while testing. [ISTQB Glossary]

[232] **Orthogonal array testing.** A systematic way of testing all-pair combinations of variables using orthogonal arrays. It significantly reduces the number of all combinations of variables to test all pair combinations. See also combinatorial testing, n-wise testing, pairwise testing. [ISTQB Glossary]

[233] «Orthogonal array», Wikipedia. [http://en.wikipedia.org/wiki/Orthogonal_array]

[234] «Orthogonal matrix», Wikipedia. [http://en.wikipedia.org/wiki/Orthogonal_matrix]

Также см. комбинаторные техники тестирования{101}, которые расширяют и дополняют только что рассмотренный список видов тестирования на основе выбора входных данных.



Крайне подробное описание некоторых видов тестирования, относящихся к данной классификации, можно найти в книге Ли Копленда «Практическое руководство по разработке тестов» (Lee Copeland, «A Practitioner's Guide to Software Test Design»), в частности:

- Тестирование на основе классов эквивалентности — в главе 3.
- Тестирование на основе граничных условий — в главе 4.
- Доменное тестирование — в главе 8.
- Попарное тестирование и тестирование на основе ортогональных массивов — в главе 6.



Важно! Большинство этих техник входит в «джентльменский набор любого тестировщика», потому их понимание и умение применять можно считать обязательным.

- Классификация на основе среды выполнения:
  - **Тестирование в процессе разработки** (development testing[235]) — тестирование, выполняемое непосредственно в процессе разработки приложения и/или в среде выполнения, отличной от среды реального использования приложения. Как правило, выполняется самими разработчиками.
  - **Операционное тестирование** (рассмотрено ранее{81}).
- **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:
  - **Тестирование по потоку управления** (control flow testing[236]) — семейство техник тестирования, в которых тест-кейсы разрабатываются с целью активации и проверки выполнения различных последовательностей событий, которые определяются посредством анализа исходного кода приложения. Дополнительное подробное пояснение см. дальше в этом разделе (см. тестирование на основе структур кода{91}).
  - **Тестирование по потоку данных** (data-flow testing[237]) — семейство техник тестирования, основанных на выборе отдельных путей из потока управления с целью исследования событий, связанных с изменением состояния переменных. Дополнительное подробное пояснение см. дальше в этом разделе (в части, где тестирование по потоку данных пояснено с точки зрения стандарта ISO/IEC/IEEE 29119-4{102}).
  - **Тестирование по диаграмме или таблице состояний** (state transition

[235] **Development testing.** Formal or informal testing conducted during the implementation of a component or system, usually in the development environment by developers. [ISTQB Glossary]

[236] **Control Flow Testing.** An approach to structure-based testing in which test cases are designed to execute specific sequences of events. Various techniques exist for control flow testing, e.g., decision testing, condition testing, and path testing, that each have their specific approach and level of control flow coverage. [ISTQB Glossary]

[237] **Data Flow Testing.** A white box test design technique in which test cases are designed to execute definition-use pairs of variables. [ISTQB Glossary]

testing[238]) — техника тестирования, в которой тест-кейсы разрабатываются для проверки переходов приложения из одного состояния в другое. Состояния могут быть описаны диаграммой состояний (state diagram[239]) или таблицей состояний (state table[240]).



Хорошее подробное пояснение по данному виду тестирования можно прочесть в статье «What is State transition testing in software testing?»[241].

Иногда эту технику тестирования также называют «тестированием по принципу конечного автомата» (finite state machine[242] 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.

- **Инспекция (аудит) кода** (code review, code inspection[243]) — 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:
  - **Тестирование на основе выражений** (statement testing[244]) — техника тестирования (по методу белого ящика), в которой проверяется корректность (и сам факт) выполнения отдельных выражений в коде.
  - **Тестирование на основе ветвей** (branch testing[245]) — 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).

[238] **State Transition Testing.** A black box test design technique in which test cases are designed to execute valid and invalid state transitions. [ISTQB Glossary]

[239] **State Diagram.** A diagram that depicts the states that a component or system can assume, and shows the events or circumstances that cause and/or result from a change from one state to another. [ISTQB Glossary]

[240] **State Table.** A grid showing the resulting transitions for each state combined with each possible event, showing both valid and invalid transitions. [ISTQB Glossary]

[241] «What is State transition testing in software testing?» [http://istqbexamcertification.com/what-is-state-transition-testing-in-software-testing/]

[242] **Finite State Machine.** A computational model consisting of a finite number of states and transitions between those states, possibly with accompanying actions. [ISTQB Glossary]

[243] **Inspection.** A type of peer review that relies on visual examination of documents to detect defects, e.g. violations of development standards and non-conformance to higher level documentation. The most formal review technique and therefore always based on a documented procedure. [ISTQB Glossary]

[244] **Statement Testing.** A white box test design technique in which test cases are designed to execute statements (statement is an entity in a programming language, which is typically the smallest indivisible unit of execution). [ISTQB Glossary]

[245] **Branch Testing.** A white box test design technique in which test cases are designed to execute branches (branch is a basic block that can be selected for execution based on a program construct in which one of two or more alternative program paths is available, e.g. case, jump, go to, if-then-else.). [ISTQB Glossary]

- **Тестирование на основе условий** (condition testing[246]) — техника тестирования (по методу белого ящика), в которой проверяется выполнение отдельных условий (условием считается выражение, которое может быть вычислено до значения «истина» или «ложь»).
- **Тестирование на основе комбинаций условий** (multiple condition testing[247]) — техника тестирования (по методу белого ящика), в которой проверяется выполнение сложных (составных) условий.
- **Тестирование на основе отдельных условий, порождающих ветвление («решающих условий»)** (modified condition decision coverage testing[248]) — 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.
- **Тестирование на основе решений** (decision testing[249]) — 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.
- **Тестирование на основе путей** (path testing[250]) — техника тестирования (по методу белого ящика), в которой проверяется выполнение всех или некоторых специально выбранных путей в коде приложения.



Если говорить строго научно, то определения большинства видов тестирования на основе структур кода должны звучать чуть-чуть иначе, т.к. в программировании условием считается выражение без логических операторов, а решением — выражение с логическими операторами. Но глоссарий ISTQB не делает на этом акцента, а потому приведённые выше определения можно считать корректными. Однако, если вам интересно, рекомендуется ознакомиться с заметкой «What is the difference between a Decision and a Condition?»[251].

Кратко вся суть видов тестирования на основе структур кода и их отличия показаны в таблице 2.3.c.

[246] **Condition Testing.** A white box test design technique in which test cases are designed to execute condition outcomes (condition is a logical expression that can be evaluated as True or False, e.g. A > B). [ISTQB Glossary]

[247] **Multiple Condition Testing.** A white box test design technique in which test cases are designed to execute combinations of single condition outcomes (within one statement). [ISTQB Glossary]

[248] **Modified Condition Decision Coverage Testing.** Technique to design test cases to execute branch condition outcomes that independently affect a decision outcome and discard conditions that do not affect the final outcome. [«Guide to Advanced Software Testing, Second Edition», Anne Mette Hass].

[249] **Decision Testing.** A white box test design technique in which test cases are designed to execute decision outcomes (decision is program point at which the control flow has two or more alternative routes, e.g. a node with two or more links to separate branches). [ISTQB Glossary]

[250] **Path testing.** A white box test design technique in which test cases are designed to execute paths. [ISTQB Glossary]

[251] «What is the difference between a Decision and a Condition?» [http://www-01.ibm.com/support/docview.wss?uid=swg21129252]

Таблица 2.3.c — Виды тестирования на основе структур кода

Русскоязычное название	Англоязычное название	Суть (что проверяется)
Тестирование на основе выражений	Statement testing	Отдельные атомарные участки кода, например x = 10
Тестирование на основе ветвей	Branch testing	Проход по ветвям выполнения кода.
Тестирование на основе условий	Condition testing, Branch Condition Testing	Отдельные условные конструкции, например if (a == b)
Тестирование на основе комбинаций условий	Multiple condition testing, Branch Condition Combination Testing	Составные условные конструкции, например if ((a == b) || (c == d))
Тестирование на основе отдельных условий, порождающих ветвление («решающих условий»)	Modified Condition Decision Coverage Testing	Отдельные условия, в одиночку влияющие на итог вычисления составного условия, например в условии if ((x == y) && (n == m)) ложное значение в каждом из отдельных условий само по себе приводит к результату false вне зависимости от результата вычисления второго условия
Тестирование на основе решений	Decision testing	Сложные ветвления, например оператор switch
Тестирование на основе путей	Path testing	Все или специально выбранные пути

- Тестирование на основе (моделей) поведения приложения (application behavior/model-based testing):
  - **Тестирование по таблице принятия решений** (decision table testing[252]) — 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.
  - **Тестирование по диаграмме или таблице состояний** (рассмотрено ранее{90}).
  - **Тестирование по спецификациям** (specification-based testing, black box testing) (рассмотрено ранее{68}).
  - **Тестирование по моделям поведения приложения** (model-based testing[253]) — техника тестирования, в которой исследование приложения (и разработка тест-кейсов) строится на некой модели: таблице принятия решений{93}, таблице или диаграмме состояний{90}, пользовательских сценариев{137}, модели нагрузки{85} и т.д.
  - **Тестирование на основе вариантов использования (**use case testing[254]) — 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

[252] **Decision Table Testing.** A black box test design technique in which test cases are designed to execute the combinations of inputs and/or stimuli (causes) shown in a decision table (a table showing combinations of inputs and/or stimuli (causes) with their associated outputs and/or actions (effects), which can be used to design test cases). [ISTQB Glossary]

[253] **Model-based Testing.** Testing based on a model of the component or system under test, e.g., reliability growth models, usage models such as operational profiles or behavioral models such as decision table or state transition diagram. [ISTQB Glossary]

[254] **Use case testing.** A black box test design technique in which test cases are designed to execute scenarios of use cases. [ISTQB Glossary]

удобно разрабатывать с помощью техник выбора входных данных{88}. В общем случае источником информации для разработки тест-кейсов в этой технике могут выступать не только варианты использования, но и другие пользовательские требования{36} 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[255]).

- **Параллельное тестирование** (parallel testing[256]) — 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{88}.
- **Тестирование на основе случайных данных** (random testing[257]) — 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[258]) — подмножеству действий, соответствующих некоей ситуации или сценарию работы с приложением. Не стоит путать этот вид тестирования с т.н. «обезьяним тестированием» (monkey testing[259]).
- **A/B-тестирование** (A/B testing, split testing[260]) — 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{82}, в которой пользователям случайным образом предлагаются разные варианты элементов интерфейса, после чего оценивается разница в реакции пользователей.



Крайне подробное описание некоторых видов тестирования, относящихся к данной классификации, можно найти в книге Ли Копленда «Практическое руководство по разработке тестов» (Lee Copeland, «A Practitioner's Guide to Software Test Design»):

- Тестирование по таблице принятия решений — в главе 5.
- Тестирование по диаграмме или таблице состояний — в главе 7.
- Тестирование на основе вариантов использования — в главе 9.

[255] **User story testing.** A black box test design technique in which test cases are designed based on user stories to verify their correct implementation. [ISTQB Glossary]

[256] **Parallel testing.** Testing a new or an altered data processing system with the same source data that is used in another system. The other system is considered as the standard of comparison. [ISPE Glossary]

[257] **Random testing.** A black box test design technique where test cases are selected, possibly using a pseudo-random generation algorithm, to match an operational profile. This technique can be used for testing non-functional attributes such as reliability and performance. [ISTQB Glossary]

[258] **Operational profile.** The representation of a distinct set of tasks performed by the component or system, possibly based on user behavior when interacting with the component or system, and their probabilities of occurrence. A task is logical rather that physical and can be executed over several machines or be executed in non-contiguous time segments. [ISTQB Glossary]

[259] **Monkey testing.** Testing by means of a random selection from a large range of inputs and by randomly pushing buttons, ignorant of how the product is being used. [ISTQB Glossary]

[260] **Split testing** is a design for establishing a causal relationship between changes and their influence on user-observable behavior. [«Controlled experiments on the web: survey and practical guide», Ron Kohav]

2.3.2.14. 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 тест-кейсов 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:
  - **Восходящее тестирование** (bottom-up testing[261]) — инкрементальный подход к интеграционному тестированию{71}, в котором в первую очередь тестируются низкоуровневые компоненты, после чего процесс переходит на всё более и более высокоуровневые компоненты.
  - **Нисходящее тестирование** (top-down testing[262]) — инкрементальный подход к интеграционному тестированию{71}, в котором в первую очередь тестируются высокоуровневые компоненты, после чего процесс переходит на всё более и более низкоуровневые компоненты.
  - **Гибридное тестирование** (hybrid testing[263]) — комбинация восходящего и нисходящего тестирования, позволяющая упростить и ускорить получение результатов оценки приложения.



Поскольку термин «гибридное» является синонимом «комбинированное», под «гибридным тестированием» может пониматься практически любое сочетание двух и более видов, техник или подходов к тестированию. Всегда уточняйте, о гибриде чего именно идёт речь.

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

[261] **Bottom-up testing.** An incremental approach to integration testing where the lowest level components are tested first, and then used to facilitate the testing of higher level components. This process is repeated until the component at the top of the hierarchy is tested. [ISTQB Glossary]

[262] **Top-down testing.** An incremental approach to integration testing where the component at the top of the component hierarchy is tested first, with lower level components being simulated by stubs. Tested components are then used to test lower level components. The process is repeated until the lowest level components have been tested. [ISTQB Glossary]

[263] **Hybrid testing, Sandwich testing.** First, the inputs for functions are integrated in the bottom-up pattern discussed above. The outputs for each function are then integrated in the top-down manner. The primary advantage of this approach is the degree of support for early release of limited functionality. [«Integration testing techniques», Kratika Parmar]

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»).
  - Типичный общий сценарий 1:
    1) Дымовое тестирование{73}.
    2) Тестирование критического пути{74}.
    3) Расширенное тестирование{75}.
  - Типичный общий сценарий 2:
    1) Модульное тестирование{71}.
    2) Интеграционное тестирование{71}.
    3) Системное тестирование{71}.
  - Типичный общий сценарий 3:
    1) Альфа-тестирование{78}.
    2) Бета-тестирование{78}.
    3) Гамма-тестирование{78}.

В завершение ещё раз подчеркнём, что рассмотренные здесь классификации тестирования не являются чем-то каноническим и незыблемым. Они лишь призваны упорядочить огромный объём информации о различных видах деятельности тестировщиков и упростить запоминание соответствующих фактов.

2.3.3. Альтернативные и дополнительные классификации тестирования

Для полноты картины остаётся лишь показать альтернативные взгляды на классификацию тестирования. Одна из них (рисунки 2.3.i и 2.3.j) представляет не более чем иную комбинацию ранее рассмотренных видов и техник. Вторая (рисунки 2.3.k и 2.3.l) содержит много новых определений, но их подробное изучение выходит за рамки данной книги, и потому будут даны лишь краткие пояснения (при необходимости вы можете ознакомиться с первоисточниками, которые указаны для каждого определения в сноске).



Ещё раз подчеркнём: здесь приведены лишь определения. Соответствующим видам и техникам тестирования в первоисточниках посвящены десятки и сотни страниц. Пожалуйста, не ожидайте от этого раздела подробных пояснений, их не будет, т.к. это — «очень дополнительный» материал.

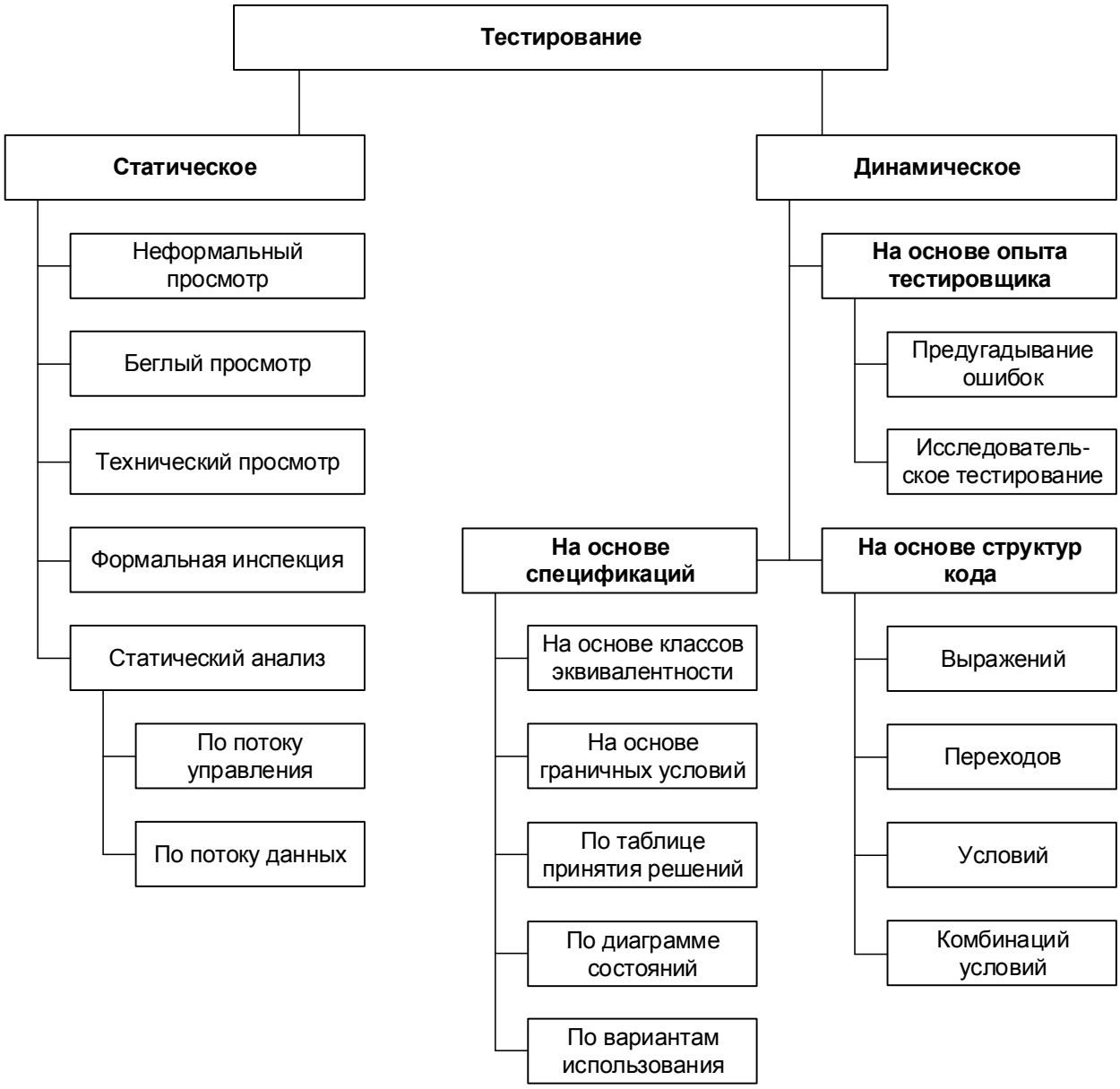


Рисунок 2.3.i — Классификация тестирования согласно книге «Foundations of Software Testing: ISTQB Certification» (Erik Van Veenendaal, Isabel Evans) (русскоязычный вариант)

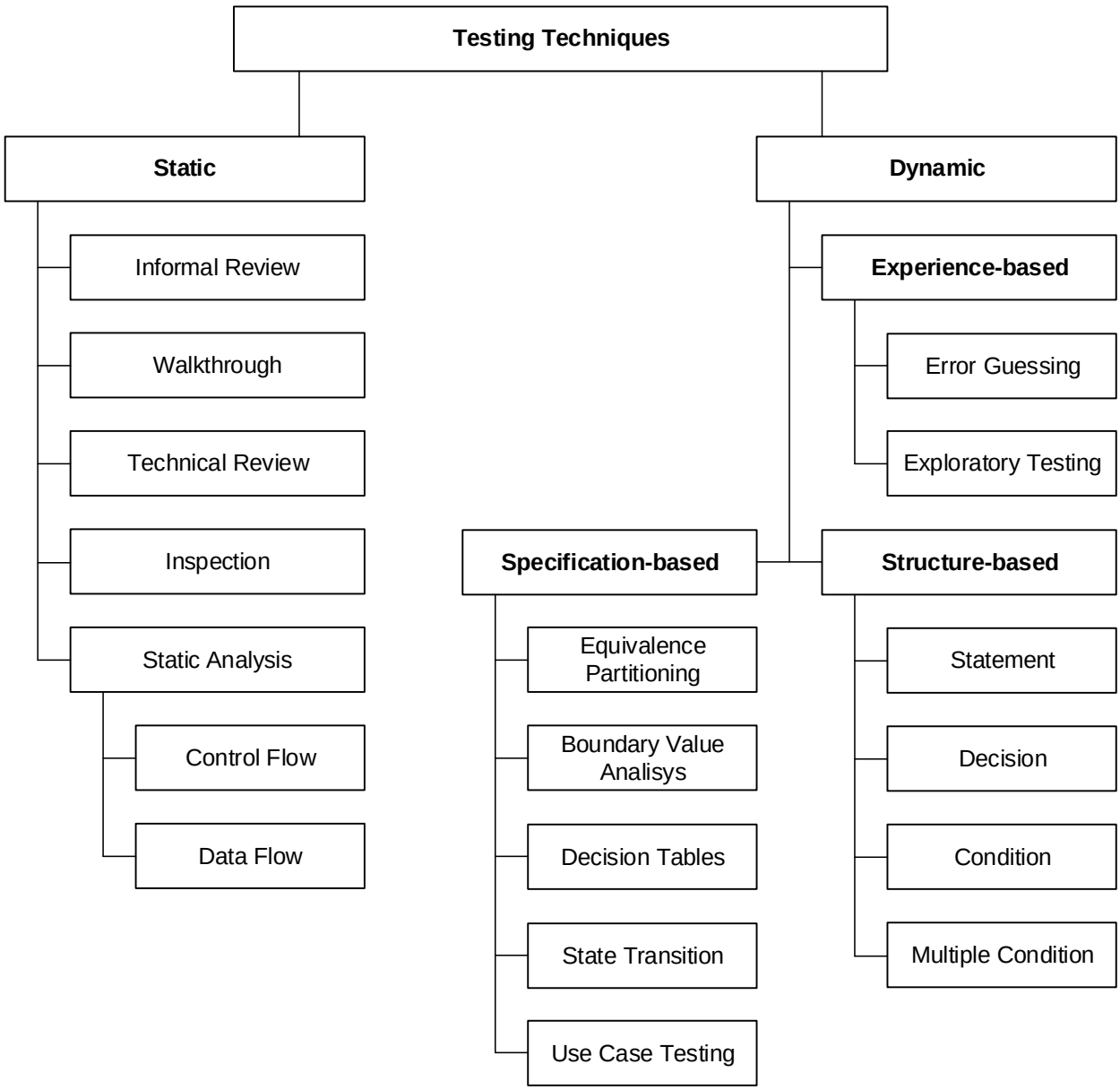


Рисунок 2.3.j — Классификация тестирования согласно книге «Foundations of Software Testing: ISTQB Certification» (Erik Van Veenendaal, Isabel Evans) (англоязычный вариант)

В следующей классификации встречаются как уже рассмотренные пункты, так и ранее не рассмотренные (отмечены пунктирной линией). Краткие определения не рассмотренных ранее видов тестирования представлены после рисунков 2.3.k и 2.3.l.

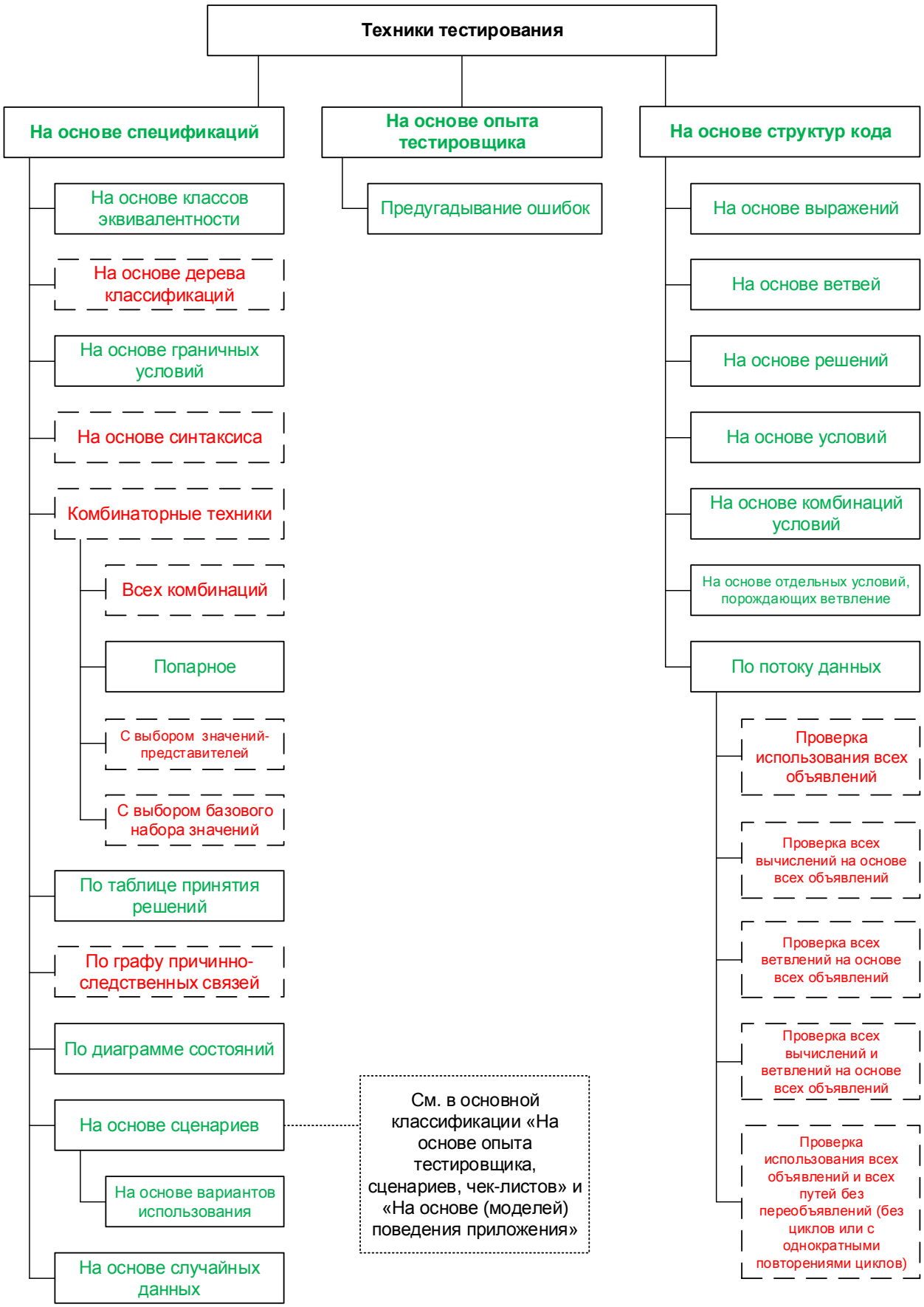


Рисунок 2.3.k — Классификация тестирования согласно ISO/IEC/IEEE 29119-4 (русскоязычный вариант)

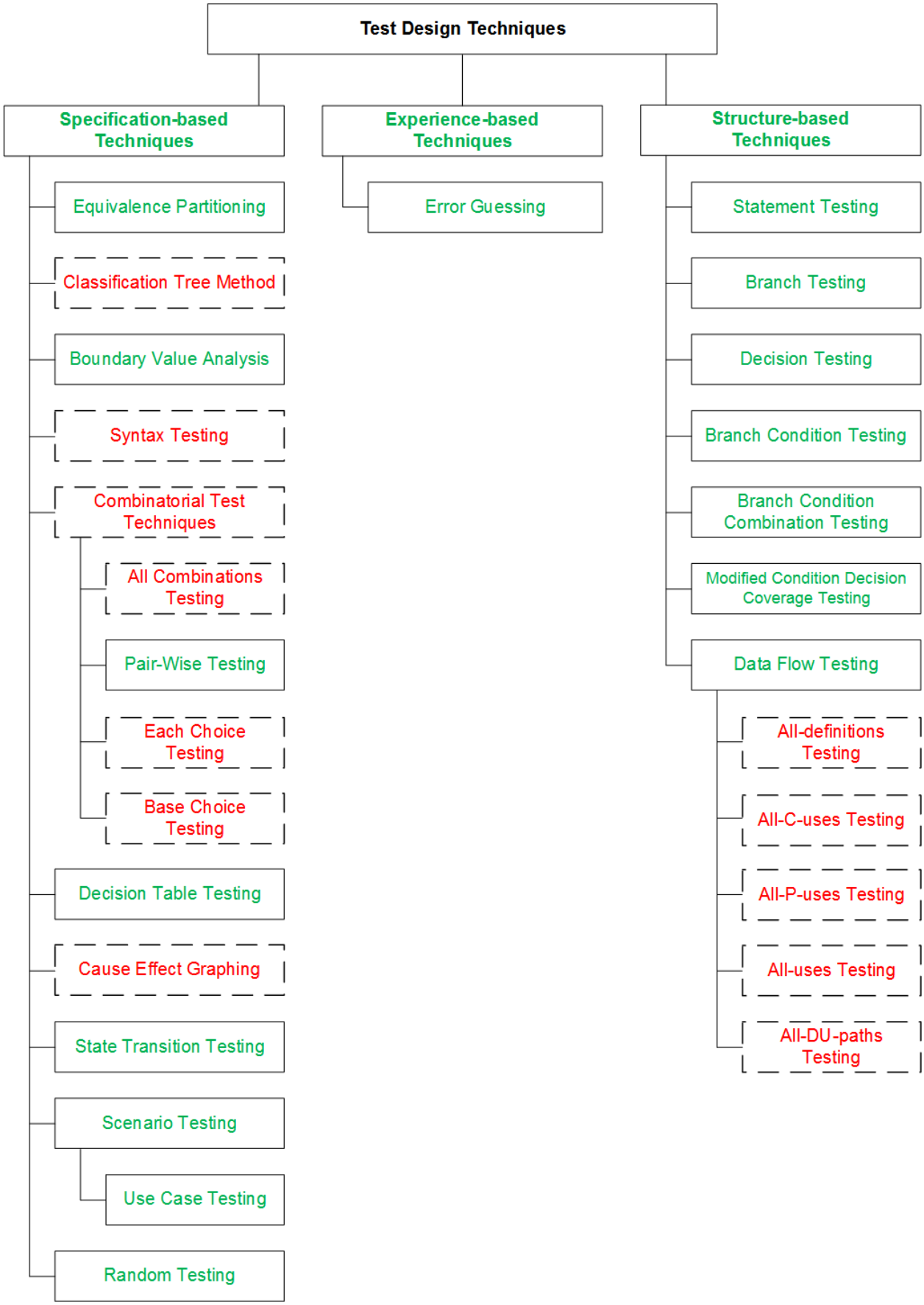


Рисунок 2.3.I — Классификация тестирования согласно ISO/IEC/IEEE 29119-4 (англоязычный вариант)

- **Тестирование на основе дерева классификаций** (classification tree[264] method[265]) — техника тестирования (по методу чёрного ящика), в которой тест-кейсы создаются на основе иерархически организованных наборов эквивалентных входных и выходных данных.
- **Тестирование на основе синтаксиса** (syntax testing[266]) — техника тестирования (по методу чёрного ящика), в которой тест-кейсы создаются на основе определения наборов входных и выходных данных.
- **Комбинаторные техники или комбинаторное тестирование** (combinatorial testing[267]) — способ выбрать подходящий набор комбинаций тестовых данных для достижения установленного уровня тестового покрытия в случае, когда проверка всех возможных наборов значений тестовых данных невозможна за имеющееся время. Существуют следующие комбинаторные техники:
  - **Тестирование всех комбинаций** (all combinations testing[268]) — тестирование всех возможных комбинаций всех значений всех тестовых данных (например, всех параметров функции).
  - **Попарное тестирование** (рассмотрено ранее{89}).
  - **Тестирование с выбором значений-представителей** (each choice testing[269]) — тестирование, при котором по одному значению из каждого набора тестовых данных должно быть использовано хотя бы в одном тест-кейсе.
  - **Тестирование с выбором базового набора значений** (base choice testing[270]) — тестирование, при котором выделяется набор значений (базовый набор), который используется для проведения тестирования в первую очередь, а далее тест-кейсы строятся на основе выбора всех базовых значений, кроме одного, которое заменяется значением, не входящим в базовый набор.

    Также см. классификацию тестирования на основе выбора входных данных{88}, которая расширяет и дополняет данный список.
- **Тестирование по графу причинно-следственных связей** (cause-effect graphing[271]) — техника тестирования (по методу чёрного ящика), в которой тест-кейсы разрабатываются на основе графа причинно-следственных связей (графического представления входных данных и воздействий со связанными с ними выходными данными и эффектами).

---

[264] **Classification tree.** A tree showing equivalence partitions hierarchically ordered, which is used to design test cases in the classification tree method. [ISTQB Glossary]

[265] **Classification tree method.** A black box test design technique in which test cases, described by means of a classification tree, are designed to execute combinations of representatives of input and/or output domains. [ISTQB Glossary]

[266] **Syntax testing.** A black box test design technique in which test cases are designed based upon the definition of the input domain and/or output domain. [ISTQB Glossary]

[267] **Combinatorial testing.** A means to identify a suitable subset of test combinations to achieve a predetermined level of coverage when testing an object with multiple parameters and where those parameters themselves each have several values, which gives rise to more combinations than are feasible to test in the time allowed. [ISTQB Glossary]

[268] **All combinations testing.** Testing of all possible combinations of all values for all parameters. [«Guide to advanced software testing, 2nd edition», Anne Matte Hass].

[269] **Each choice testing.** One value from each block for each partition must be used in at least one test case. [«Introduction to Software Testing. Chapter 4. Input Space Partition Testing», Paul Ammann & Jeff Offutt]

[270] **Base choice testing.** A base choice block is chosen for each partition, and a base test is formed by using the base choice for each partition. Subsequent tests are chosen by holding all but one base choice constant and using each non-base choice in each other parameter. [«Introduction to Software Testing. Chapter 4. Input Space Partition Testing», Paul Ammann & Jeff Offutt]

[271] **Cause-effect graphing.** A black box test design technique in which test cases are designed from cause-effect graphs (a graphical representation of inputs and/or stimuli (causes) with their associated outputs (effects), which can be used to design test cases). [ISTQB Glossary]

- **Тестирование по потоку данных** (data-flow testing[272]) — семейство техник тестирования, основанных на выборе отдельных путей из потока управления с целью исследования событий, связанных с изменением состояния переменных. Эти техники позволяют обнаружить такие ситуации, как: переменная определена, но нигде не используется; переменная используется, но не определена; переменная определена несколько раз до того, как она используется; переменная удалена до последнего случая использования.

  Здесь придётся немного погрузиться в теорию. Над переменной в общем случае может выполняться несколько действий (покажем на примере переменной **x**):
  - объявление (declaration): int **x**;
  - определение (definition, d-use): **x** = 99;
  - использование в вычислениях (computation use, c-use): z = **x** + 1;
  - использование в условиях (predicate use, p-use): if (**x** > 17) { … };
  - удаление (kill, k-use): **x** = null;

  Теперь можно рассмотреть техники тестирования на основе потока данных. Они крайне подробно описаны в разделе 3.3 главы 5 книги Бориса Бейзера «Техники тестирования ПО» («Software Testing Techniques, Second Edition», Boris Beizer), мы же приведём очень краткие пояснения:
  - **Проверка использования всех объявлений** (all-definitions testing[273]) — тестовым набором проверяется, что для каждой переменной существует путь от её определения к её использованию в вычислениях или условиях.
  - **Проверка всех вычислений на основе всех объявлений** (all-c-uses testing[274]) — тестовым набором проверяется, что для каждой переменной существует путь от каждого её определения к её использованию в вычислениях.
  - **Проверка всех ветвлений на основе всех объявлений** (all-p-uses testing[275]) — тестовым набором проверяется, что для каждой переменной существует путь от каждого её определения к её использованию в условиях.
  - **Проверка всех вычислений и ветвлений на основе всех объявлений** (all-uses testing[276]) — тестовым набором проверяется, что для каждой переменной существует хотя бы один путь от каждого её определения к каждому её использованию в вычислениях и в условиях.
  - **Проверка использования всех объявлений и всех путей без переобъявлений (без циклов или с однократными повторениями циклов)** (all-du-paths testing[277]) — тестовым набором для каждой переменной проверяются все пути от каждого её определения к каждому её использованию в вычислениях и в условиях (самая мощная стратегия, которая в то же время требует наибольшего количества тест-кейсов).

[272] **Data flow testing.** A white box test design technique in which test cases are designed to execute definition-use pairs of variables. [ISTQB Glossary]

[273] **All-definitions strategy.** Test set requires that every definition of every variable is covered by at least one use of that variable (c-use or p-use). [«Software Testing Techniques, Second Edition», Boris Beizer]

[274] **All-computation-uses strategy.** For every variable and every definition of that variable, include at least one definition-free path from the definition to every computation use. [«Software Testing Techniques, Second Edition», Boris Beizer]

[275] **All-predicate-uses strategy.** For every variable and every definition of that variable, include at least one definition-free path from the definition to every predicate use. [«Software Testing Techniques, Second Edition», Boris Beizer]

[276] **All-uses strategy.** Test set includes at least one path segment from every definition to every use that can be reached by that definition. [«Software Testing Techniques, Second Edition», Boris Beizer]

[277] **All-DU-path strategy.** Test set includes every du path from every definition of every variable to every use of that definition. [«Software Testing Techniques, Second Edition», Boris Beizer]

Для лучшего понимания и запоминания приведём оригинальную схему из книги Бориса Бейзера (там она фигурирует под именем «Figure 5.7. Relative Strength of Structural Test Strategies»), показывающую взаимосвязь стратегий тестирования на основе потока данных (рисунок 2.3.m).

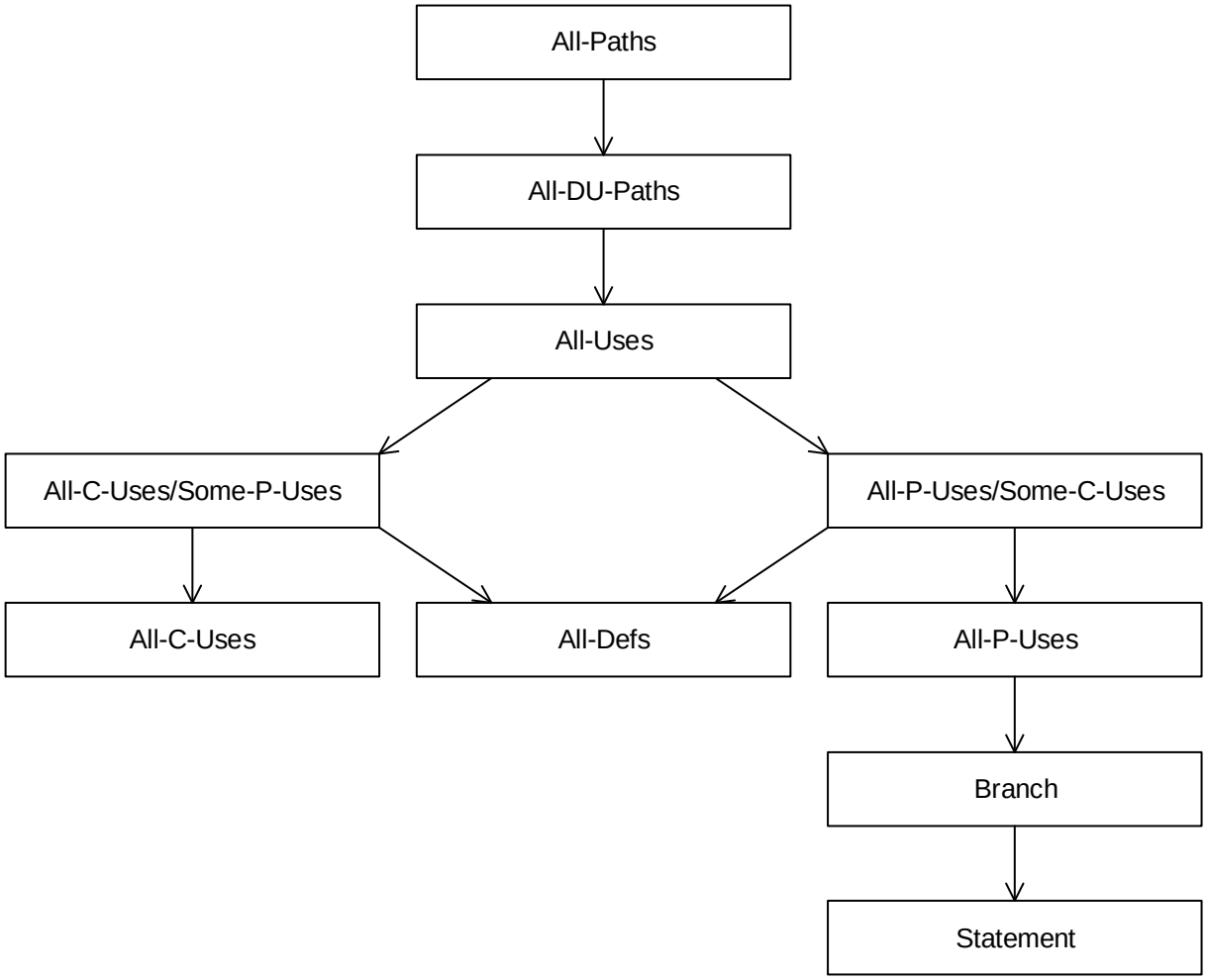


Рисунок 2.3.m — Взаимосвязь и относительная мощность стратегий тестирования на основе потока данных (по книге Бориса Бейзера «Техники тестирования ПО»)

2.3.4. Классификация по принадлежности к тестированию по методу белого и чёрного ящиков

Типичнейшим вопросом на собеседовании для начинающих тестировщиков является просьба перечислить техники тестирования по методу белого и чёрного ящиков. Ниже представлена таблица 2.3.d, в которой все вышерассмотренные виды тестирования соотнесены с соответствующим методом. Эту таблицу можно использовать также как справочник по видам тестирования (они представлены в той же последовательности, в какой описаны в данной главе).



Важно! В источниках наподобие ISTQB-глоссария многие виды и техники тестирования жёстко соотнесены с методами белого или чёрного ящика. Но это не значит, что их невозможно применить в другом, не отмеченном методе. Так, например, тестирование на основе классов эквивалентности отнесено к методу чёрного ящика, но оно прекрасно подходит и для написания модульных тест-кейсов, являющихся ярчайшими представителями тестирования по методу белого ящика.

Воспринимайте данные из представленной ниже таблицы не как «этот вид тестирования может применяться только для…», а как «чаще всего этот вид тестирования применяется для…»

Таблица 2.3.d — Виды и техники тестирования в контексте методов белого и чёрного ящиков

Вид тестирования (русскоязычное название)	Вид тестирования (англоязычное название)	Белый ящик	Чёрный ящик
Статическое тестирование{67}	Static testing	Да	Нет
Динамическое тестирование{67}	Dynamic testing	Изредка	Да
Ручное тестирование{69}	Manual testing	Мало	Да
Автоматизированное тестирование{69}	Automated testing	Да	Да
Модульное (компонентное) тестирование{71}	Unit testing, Module testing, Component testing	Да	Нет
Интеграционное тестирование{71}	Integration testing	Да	Да
Системное тестирование{71}	System testing	Мало	Да
Дымовое тестирование{73}	Smoke test, Intake test, Build verification test	Мало	Да
Тестирование критического пути{74}	Critical path test	Мало	Да
Расширенное тестирование{75}	Extended test	Мало	Да
Позитивное тестирование{76}	Positive testing	Да	Да
Негативное тестирование{76}	Negative testing, Invalid testing	Да	Да
Тестирование веб-приложений{77}	Web-applications testing	Да	Да
Тестирование мобильных приложений{77}	Mobile applications testing	Да	Да
Тестирование настольных приложений{77}	Desktop applications testing	Да	Да

Тестирование уровня представления[77]	Presentation tier testing	Мало	Да
Тестирование уровня бизнес-логики[77]	Business logic tier testing	Да	Да
Тестирование уровня данных[77]	Data tier testing	Да	Мало
Альфа-тестирование[78]	Alpha testing	Мало	Да
Бета-тестирование[78]	Beta testing	Почти никогда	Да
Гамма-тестирование[78]	Gamma testing	Почти никогда	Да
Тестирование на основе тест-кейсов[78]	Scripted testing, Test case based testing	Да	Да
Исследовательское тестирование[78]	Exploratory testing	Нет	Да
Свободное (интуитивное) тестирование[79]	Ad hoc testing	Нет	Да
Функциональное тестирование[79]	Functional testing	Да	Да
Нефункциональное тестирование[80]	Non-functional testing	Да	Да
Инсталляционное тестирование[80]	Installation testing	Изредка	Да
Регрессионное тестирование[80]	Regression testing	Да	Да
Повторное тестирование[81]	Re-testing, Confirmation testing	Да	Да
Приёмочное тестирование[81]	Acceptance testing	Крайне редко	Да
Операционное тестирование[81]	Operational testing	Крайне редко	Да
Тестирование удобства использования[82]	Usability testing	Крайне редко	Да
Тестирование доступности[82]	Accessibility testing	Крайне редко	Да
Тестирование интерфейса[82]	Interface testing	Да	Да
Тестирование безопасности[83]	Security testing	Да	Да
Тестирование интернационализации[83]	Internationalization testing	Мало	Да
Тестирование локализации[83]	Localization testing	Мало	Да
Тестирование совместимости[83]	Compatibility testing	Мало	Да
Конфигурационное тестирование[83]	Configuration testing	Мало	Да
Кросс-браузерное тестирование[83]	Cross-browser testing	Мало	Да
Тестирование данных и баз данных[84]	Data quality testing and Database integrity testing	Да	Мало
Тестирование использования ресурсов[84]	Resource utilization testing	Крайне редко	Да
Сравнительное тестирование[84]	Comparison testing	Нет	Да

Демонстрационное тестирование[85]	Qualification testing	Нет	Да
Избыточное тестирование[85]	Exhaustive testing	Крайне редко	Нет
Тестирование надёжности[85]	Reliability testing	Крайне редко	Да
Тестирование восстанавливаемости[85]	Recoverability testing	Крайне редко	Да
Тестирование отказоустойчивости[85]	Failover testing	Крайне редко	Да
Тестирование производительности[85]	Performance testing	Крайне редко	Да
Нагрузочное тестирование[85]	Load testing, Capacity testing	Крайне редко	Да
Тестирование масштабируемости[86]	Scalability testing	Крайне редко	Да
Объёмное тестирование[86]	Volume testing	Крайне редко	Да
Стрессовое тестирование[86]	Stress testing	Крайне редко	Да
Конкурентное тестирование[86]	Concurrency testing	Крайне редко	Да
Инвазивное тестирование[87]	Intrusive testing	Да	Да
Неинвазивное тестирование[87]	Nonintrusive testing	Да	Да
Тестирование под управлением данными[87]	Data-driven testing	Да	Да
Тестирование под управлением ключевыми словами[87]	Keyword-driven testing	Да	Да
Тестирование предугадыванием ошибок[88]	Error guessing	Крайне редко	Да
Эвристическая оценка[88]	Heuristic evaluation	Нет	Да
Мутационное тестирование[88]	Mutation testing	Да	Да
Тестирование добавлением ошибок[88]	Error seeding	Да	Да
Тестирование на основе классов эквивалентности[88]	Equivalence partitioning	Да	Да
Тестирование на основе граничных условий[89]	Boundary value analysis	Да	Да
Доменное тестирование[89]	Domain testing, Domain analysis	Да	Да
Попарное тестирование[89]	Pairwise testing	Да	Да
Тестирование на основе ортогональных массивов[89]	Orthogonal array testing	Да	Да
Тестирование в процессе разработки[90]	Development testing	Да	Да
Тестирование по потоку управления[90]	Control flow testing	Да	Нет
Тестирование по потоку данных[90]	Data flow testing	Да	Нет
Тестирование по диаграмме или таблице состояний[90]	State transition testing	Да	Нет
Инспекция (аудит) кода[91]	Code review, code inspection	Да	Нет

Тестирование на основе выражений[91]	Statement testing	Да	Нет
Тестирование на основе ветвей[91]	Branch testing	Да	Нет
Тестирование на основе условий[92]	Condition testing	Да	Нет
Тестирование на основе комбинаций условий[92]	Multiple condition testing	Да	Нет
Тестирование на основе отдельных условий, порождающих ветвление[92] («решающих условий»)	Modified condition decision coverage testing	Да	Нет
Тестирование на основе решений[92]	Decision testing	Да	Нет
Тестирование на основе путей[92]	Path testing	Да	Нет
Тестирование по таблице принятия решений[93]	Decision table testing	Да	Да
Тестирование по моделям поведения приложения[93]	Model-based testing	Да	Да
Тестирование на основе вариантов использования[93]	Use case testing	Да	Да
Параллельное тестирование[94]	Parallel testing	Да	Да
Тестирование на основе случайных данных[94]	Random testing	Да	Да
A/B-тестирование[94]	A/B testing, Split testing	Нет	Да
Восходящее тестирование[95]	Bottom-up testing	Да	Да
Нисходящее тестирование[95]	Top-down testing	Да	Да
Гибридное тестирование[95]	Hybrid testing	Да	Да
Тестирование на основе дерева классификаций[101]	Classification tree method	Да	Да
Тестирование на основе синтаксиса[101]	Syntax testing	Да	Да
Комбинаторные техники[101] (комбинаторное тестирование)	Combinatorial testing	Да	Да
Тестирование всех комбинаций[101]	All combinations testing	Да	Нет
Тестирование с выбором значений-представителей[101]	Each choice testing	Да	Нет
Тестирование с выбором базового набора значений[101]	Base choice testing	Да	Нет
Тестирование по графу причинно-следственных связей[101]	Cause-effect graphing	Да	Нет
Проверка использования всех объявлений[102]	All-definitions testing	Да	Нет

Проверка всех вычислений на основе всех объявлений[102]	All-c-uses testing	Да	Нет
Проверка всех ветвлений на основе всех объявлений[102]	All-p-uses testing	Да	Нет
Проверка всех вычислений и ветвлений на основе всех объявлений[102]	All-uses testing	Да	Нет
Проверка использования всех объявлений и всех путей без переобъявлений[102] (без циклов или с однократными повторениями циклов)	All-du-paths testing	Да	Нет