Тестировщика должны всему-всему научить

Не должны. И уж тем более «всему-всему». Да, если мы говорим о явно обозначенном учебном процессе, то его организаторы (будь то предмет в университете, учебный курс в некоем тренинговом центре или отдельный тренинг внутри компании) часто берут на себя определённые «педагогические обязательства». Но подобная учебная деятельность никогда не заменит саморазвития (хотя и может в нужный момент помочь в выборе направления пути). IT-отрасль меняется очень интенсивно и непрерывно. Учить**ся** придётся до пенсии.

В тестировщики идут те, кто не смог стать программистом

А в скрипачи — те, кто не смог стать пианистом? Я думаю, что некий небольшой процент «не ставших программистами» в тестировании есть. Но он теряется на фоне тех, кто шёл в тестирование изначально и сознательно, а также тех, кто пришёл в тестирование из программирования.

В тестировании сложно построить карьеру

При должном старании карьера в тестировании оказывается едва ли не самой динамичной (по сравнению с другими IT-направлениями). Тестирование само по себе — очень бурно развивающаяся отрасль IT, и здесь всегда можно выбрать что-то, что будет вам очень нравиться и хорошо получаться — а в таких условиях стать профессионалом и достичь успеха легко.

Тестировщик «виноват во всём», т.е. с него спрос за все ошибки

Только если признать, что в болезни пациента виновен термометр, показывающий высокую температуру. Скорее с тестировщиков будет спрос за те ошибки, что были найдены пользователем, т.е. проявились уже на стадии реальной эксплуатации продукта. Но и здесь нет однозначного вывода — за конечный успех продукта отвечает вся команда, и было бы глупо перекладывать ответственность лишь на одну её часть.

Тестировщики скоро будут не нужны, т.к. всё будет автоматизировано

Как только по улицам забегают терминаторы — да, этот миф станет правдой: программы научатся обходиться без людей. Но тогда у нас всех будут другие проблемы. А если кроме шуток, человечество уже сотни лет идёт по пути автоматизации, которая накладывает свой отпечаток на всю нашу жизнь и чаще всего позволяет переложить самую простую и не требующую квалификации работу на машины. Но кто же заставляет вас оставаться на уровне исполнителя такой работы? Начиная с некоторого уровня, тестирование превращается в гармоничное сочетание науки и искусства. А многих ли учёных или творцов заменила автоматизация?

Просьба: возможно, у вас есть какие-то мысли из разряда «А я думал(а), что в тестировании…» / «А правда ли, что в тестировании…». Если так, поделитесь ими, пожалуйста, в анонимном опросе: http://svyatoslav.biz/software_testing_book_poll/

Раздел 2: основные знания и умения

2.1. Процессы тестирования и разработки ПО

2.1.1. Модели разработки ПО

Чтобы лучше разобраться в том, как тестирование соотносится с программированием и иными видами проектной деятельности, для начала рассмотрим самые основы — модели разработки (lifecycle model[15]) ПО (как часть жизненного цикла (software lifecycle[16]) ПО). При этом сразу подчеркнём, что разработка ПО является лишь частью жизненного цикла ПО, и здесь мы говорим именно о **разработке**.

Материал данной главы относится скорее к дисциплине «управление проектами», потому здесь рассмотрен крайне сжато: пожалуйста, не воспринимайте его как исчерпывающее руководство — здесь едва ли рассмотрена и сотая доля процента соответствующей предметной области.

Модель разработки ПО (Software Development Model, SDM) — структура, систематизирующая различные виды проектной деятельности, их взаимодействие и последовательность в процессе разработки ПО. Выбор той или иной модели зависит от масштаба и сложности проекта, предметной области, доступных ресурсов и множества других факторов.

Выбор модели разработки ПО серьёзно влияет на процесс тестирования, определяя выбор стратегии, расписание, необходимые ресурсы и т.д.

Моделей разработки ПО много, но в общем случае классическими можно считать водопадную, v-образную, итерационную инкрементальную, спиральную и гибкую.

Перечень моделей разработки ПО (с кратким описанием), рекомендуемых к изучению тестировщиками, можно найти в статье «What are the Software Development Models?»[17].

Знать и понимать модели разработки ПО необходимо затем, чтобы уже с первых дней работы понимать, что происходит вокруг, что, зачем и почему вы делаете. Многие начинающие тестировщики отмечают, что ощущение бессмысленности происходящего посещает их, даже если текущие задания интересны. Чем полнее вы будете представлять картину происходящего на проекте, тем яснее вам будет виден ваш собственный вклад в общее дело и смысл того, чем вы занимаетесь.

Ещё одна важная вещь, которую следует понимать, состоит в том, что никакая модель не является догмой или универсальным решением. Нет идеальной модели. Есть та, которая хуже или лучше подходит для конкретного проекта, конкретной команды, конкретных условий.

Частая ошибка! Единственное, от чего стоит предостеречь уже сейчас, так это от фривольной трактовки модели и перекраивания её «на свой вкус» без кристально чёткого понимания, что и зачем вы делаете. О том, что бывает при нарушении логики модели, прекрасно сказал в своём слайдкасте «Scrum Tailoring»[18] Максим Дорофеев.

---

[15] **Lifecycle model.** A partitioning of the life of a product or project into phases. [ISTQB Glossary]

[16] **Software lifecycle.** The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software lifecycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and sometimes, retirement phase. Note these phases may overlap or be performed iteratively. [ISTQB Glossary]

[17] «What are the Software Development Models?» [http://istqbexamcertification.com/what-are-the-software-development-models/]

[18] «Scrum Tailoring», Максим Дорофеев [http://cartmendum.livejournal.com/10862.html]

Водопадная модель (waterfall model[19]) сейчас представляет скорее исторический интерес, т.к. в современных проектах практически неприменима. Она предполагает однократное выполнение каждой из фаз проекта, которые, в свою очередь, строго следуют друг за другом (рисунок 2.1.a). Очень упрощённо можно сказать, что в рамках этой модели в любой момент времени команде «видна» лишь предыдущая и следующая фаза. В реальной же разработке ПО приходится «видеть весь проект целиком» и возвращаться к предыдущим фазам, чтобы исправить недоработки или что-то уточнить.

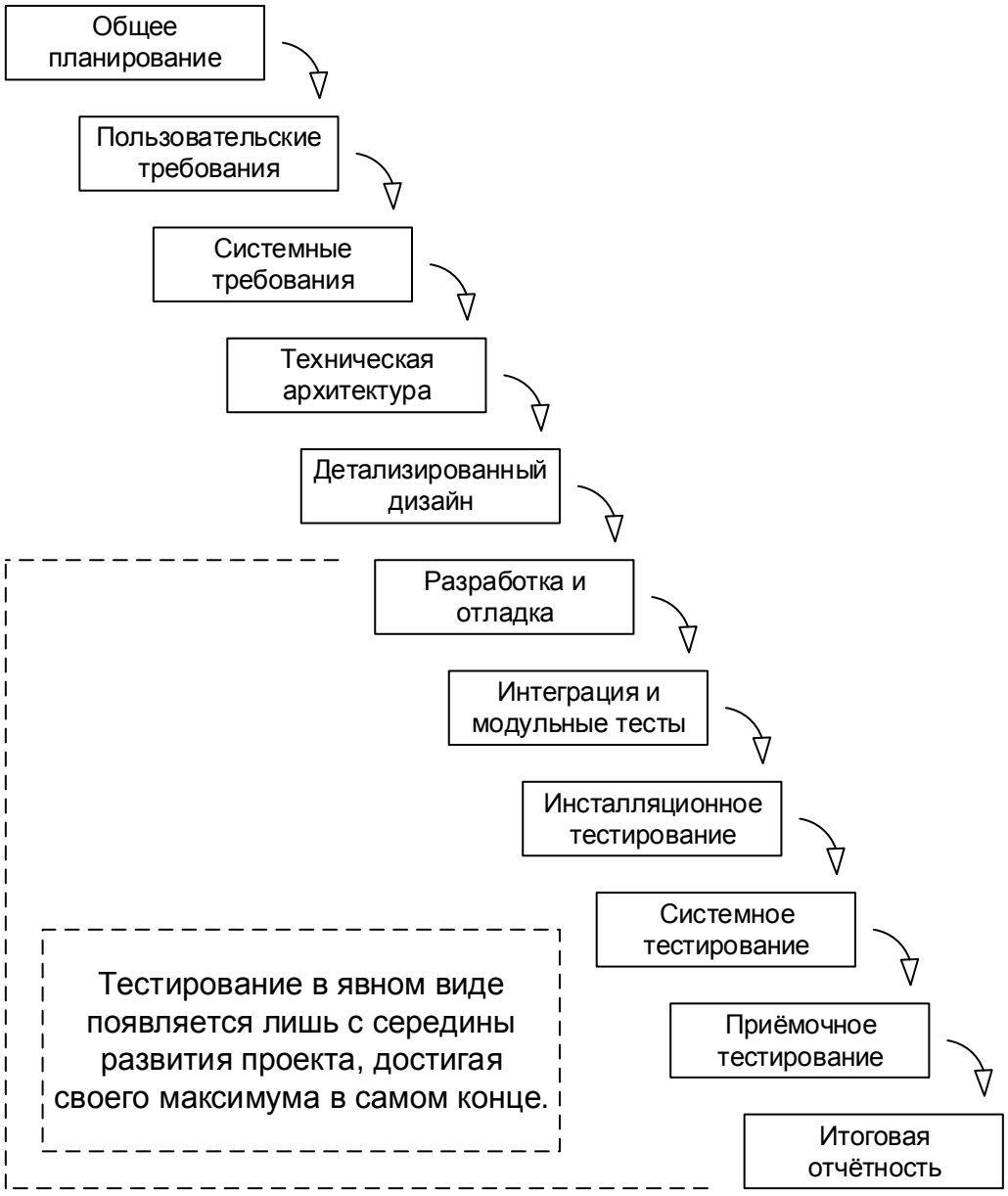

Рисунок 2.1.a — Водопадная модель разработки ПО

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

[19] In a **waterfall model**, each phase must be completed fully before the next phase can begin. This type of model is basically used for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. [http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/]

Тем не менее водопадная модель часто интуитивно применяется при выполнении относительно простых задач, а её недостатки послужили прекрасным отправным пунктом для создания новых моделей. Также эта модель в несколько усовершенствованном виде используется на крупных проектах, в которых требования очень стабильны и могут быть хорошо сформулированы в начале проекта (аэрокосмическая область, медицинское ПО и т.д.)

Относительно краткое и притом хорошее описание водопадной модели можно найти в статье «What is Waterfall model advantages, disadvantages and when to use it?»[20].

Великолепное описание истории развития и заката водопадной модели было создано Максимом Дорофеевым в виде слайдкаста «The Rise And Fall Of Waterfall», который можно посмотреть[21] в его ЖЖ.

V-образная модель (V-model[22]) является логическим развитием водопадной. Можно заметить (рисунок 2.1.b), что в общем случае как водопадная, так и v-образная модели жизненного цикла ПО могут содержать один и тот же набор стадий, но принципиальное отличие заключается в том, как эта информация используется в процессе реализации проекта.

Очень упрощённо можно сказать, что при использовании v-образной модели на каждой стадии «на спуске» нужно думать о том, что и как будет происходить на соответствующей стадии «на подъёме». Тестирование здесь появляется уже на самых ранних стадиях развития проекта, что позволяет минимизировать риски, а также обнаружить и устранить множество потенциальных проблем до того, как они станут проблемами реальными.

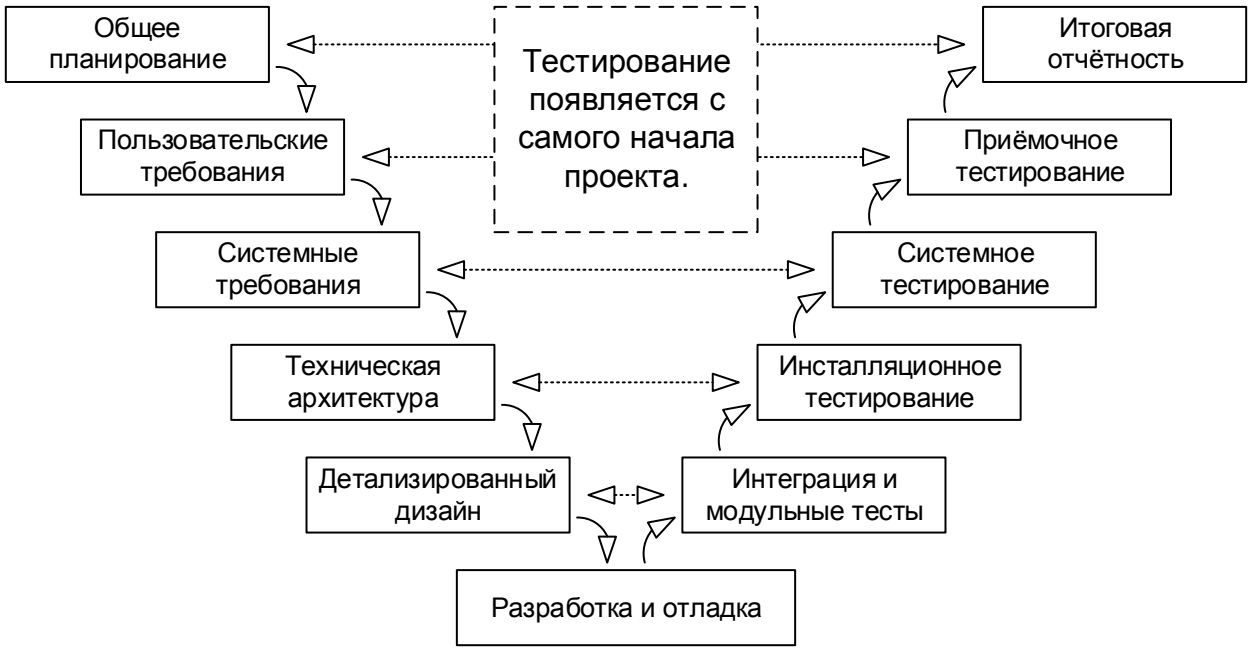

Рисунок 2.1.b — V-образная модель разработки ПО

[20] «What is Waterfall model advantages, disadvantages and when to use it?» [http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/]

[21] ЖЖ Максима Дорофеева. [http://cartmendum.livejournal.com/44064.html]

[22] **V-model.** A framework to describe the software development lifecycle activities from requirements specification to maintenance. The V-model illustrates how testing activities can be integrated into each phase of the software development lifecycle. [ISTQB Glossary]

Краткое описание v-образной модели можно найти в статье «What is V-model advantages, disadvantages and when to use it?»[23]. Пояснение по использованию v-образной модели в тестировании можно найти в статье «Using V Models for Testing»[24].

Итерационная инкрементальная модель (iterative model[25], incremental model[26]) является фундаментальной основой современного подхода к разработке ПО. Как следует из названия модели, ей свойственна определённая двойственность (а ISTQB-глоссарий даже не приводит единого определения, разбивая его на отдельные части):

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

Ключевой особенностью данной модели является разбиение проекта на относительно небольшие промежутки (итерации), каждый из которых в общем случае может включать в себя все классические стадии, присущие водопадной и v-образной моделям (рисунок 2.1.c). Итогом итерации является приращение (инкремент) функциональности продукта, выраженное в промежуточном билде (build[27]).

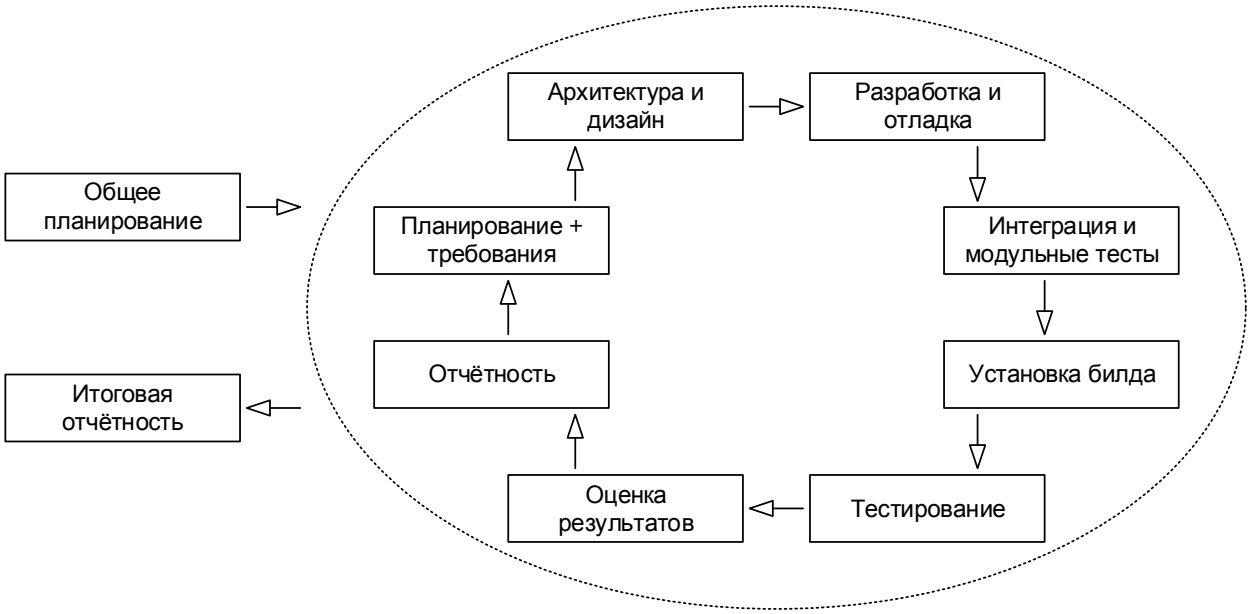

Рисунок 2.1.c — Итерационная инкрементальная модель разработки ПО

[23] «What is V-model advantages, disadvantages and when to use it?» [http://istqbexamcertification.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/]

[24] «Using V Models for Testing», Donald Firesmith [http://blog.sei.cmu.edu/post.cfm/using-v-models-testing-315]

[25] **Iterative development model.** A development lifecycle where a project is broken into a usually large number of iterations. An iteration is a complete development loop resulting in a release (internal or external) of an executable product, a subset of the final product under development, which grows from iteration to iteration to become the final product. [ISTQB Glossary]

[26] **Incremental development model.** A development lifecycle where a project is broken into a series of increments, each of which delivers a portion of the functionality in the overall project requirements. The requirements are prioritized and delivered in priority order in the appropriate increment. In some (but not all) versions of this lifecycle model, each subproject follows a 'mini V-model' with its own design, coding and testing phases. [ISTQB Glossary]

[27] **Build.** A development activity whereby a complete system is compiled and linked, so that a consistent system is available including all latest changes. [На основе определения термина «daily build» из ISTQB Glossary]

Длина итераций может меняться в зависимости от множества факторов, однако сам принцип многократного повторения позволяет гарантировать, что и тестирование, и демонстрация продукта конечному заказчику (с получением обратной связи) будет активно применяться с самого начала и на протяжении всего времени разработки проекта.

Во многих случаях допускается распараллеливание отдельных стадий внутри итерации и активная доработка с целью устранения недостатков, обнаруженных на любой из (предыдущих) стадий.

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

Относительно краткие и очень хорошие описания итерационной инкрементальной модели можно найти в статьях «What is Iterative model advantages, disadvantages and when to use it?»[28] и «What is Incremental model advantages, disadvantages and when to use it?»[29].

Спиральная модель (spiral model[30]) представляет собой частный случай итерационной инкрементальной модели, в котором особое внимание уделяется управлению рисками, в особенности влияющими на организацию процесса разработки проекта и контрольные точки.

Схематично суть спиральной модели представлена на рисунке 2.1.d. Обратите внимание на то, что здесь явно выделены четыре ключевые фазы:

- проработка целей, альтернатив и ограничений;
- анализ рисков и прототипирование;
- разработка (промежуточной версии) продукта;
- планирование следующего цикла.

С точки зрения тестирования и управления качеством повышенное внимание рискам является ощутимым преимуществом при использовании спиральной модели для разработки концептуальных проектов, в которых требования естественным образом являются сложными и нестабильными (могут многократно меняться по ходу выполнения проекта).

Автор модели Barry Boehm в своих публикациях[31, 32] подробно раскрывает эти вопросы и приводит множество рассуждений и рекомендаций о том, как применять спиральную модель с максимальным эффектом.

Относительно краткие и очень хорошие описания спиральной модели можно найти в статьях «What is Spiral model- advantages, disadvantages and when to use it?»[33] и «Spiral Model»[34].

---

[28] «What is Iterative model advantages, disadvantages and when to use it?» [http://istqbexamcertification.com/what-is-iterative-model-advantages-disadvantages-and-when-to-use-it/]

[29] «What is Incremental model advantages, disadvantages and when to use it?» [http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/]

[30] **Spiral model.** A software lifecycle model which supposes incremental development, using the waterfall model for each step, with the aim of managing risk. In the spiral model, developers define and implement features in order of decreasing priority. [http://dictionary.reference.com/browse/spiral+model]

[31] «A Spiral Model of Software Development and Enhancement», Barry Boehm [http://csse.usc.edu/csse/TECHRPTS/1988/usccse88-500/usccse88-500.pdf]

[32] «Spiral Development: Experience, Principles, and Refinements», Barry Boehm. [http://www.sei.cmu.edu/reports/00sr008.pdf]

[33] «What is Spiral model- advantages, disadvantages and when to use it?» [http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/]

[34] «Spiral Model», Amir Ghahrai. [http://www.testingexcellence.com/spiral-model/]

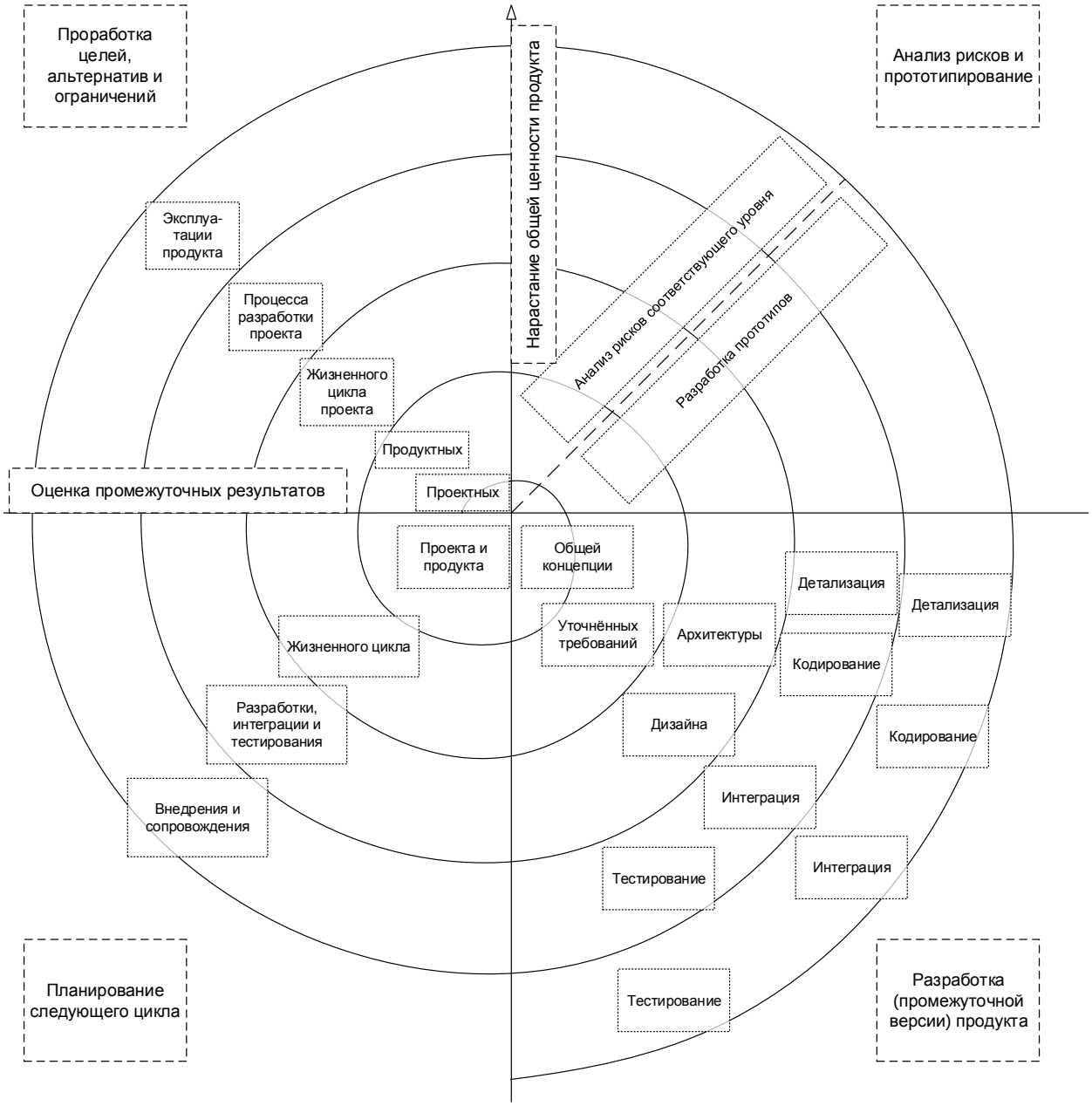

Рисунок 2.1.d — Спиральная модель разработки ПО

Гибкая модель (agile model[35]) представляет собой совокупность различных подходов к разработке ПО и базируется на т.н. «agile-манифесте»[36]:

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

Данная тема является настолько большой, что ссылок на статьи недостаточно, а потому стоит почитать эти книги:

- «Agile Testing» (Lisa Crispin, Janet Gregory).
- «Essential Scrum» (Kenneth S. Rubin).

[35] **Agile software development.** A group of software development methodologies based on EITP iterative incremental development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. [ISTQB Glossary]

[36] «Agile-манифест» [http://agilemanifesto.org/iso/ru/manifesto.html]

Как несложно догадаться, положенные в основу гибкой модели подходы являются логическим развитием и продолжением всего того, что было за десятилетия создано и опробовано в водопадной, v-образной, итерационной инкрементальной, спиральной и иных моделях. Причём здесь впервые был достигнут ощутимый результат в снижении бюрократической составляющей и максимальной адаптации процесса разработки ПО к мгновенным изменениям рынка и требований заказчика.

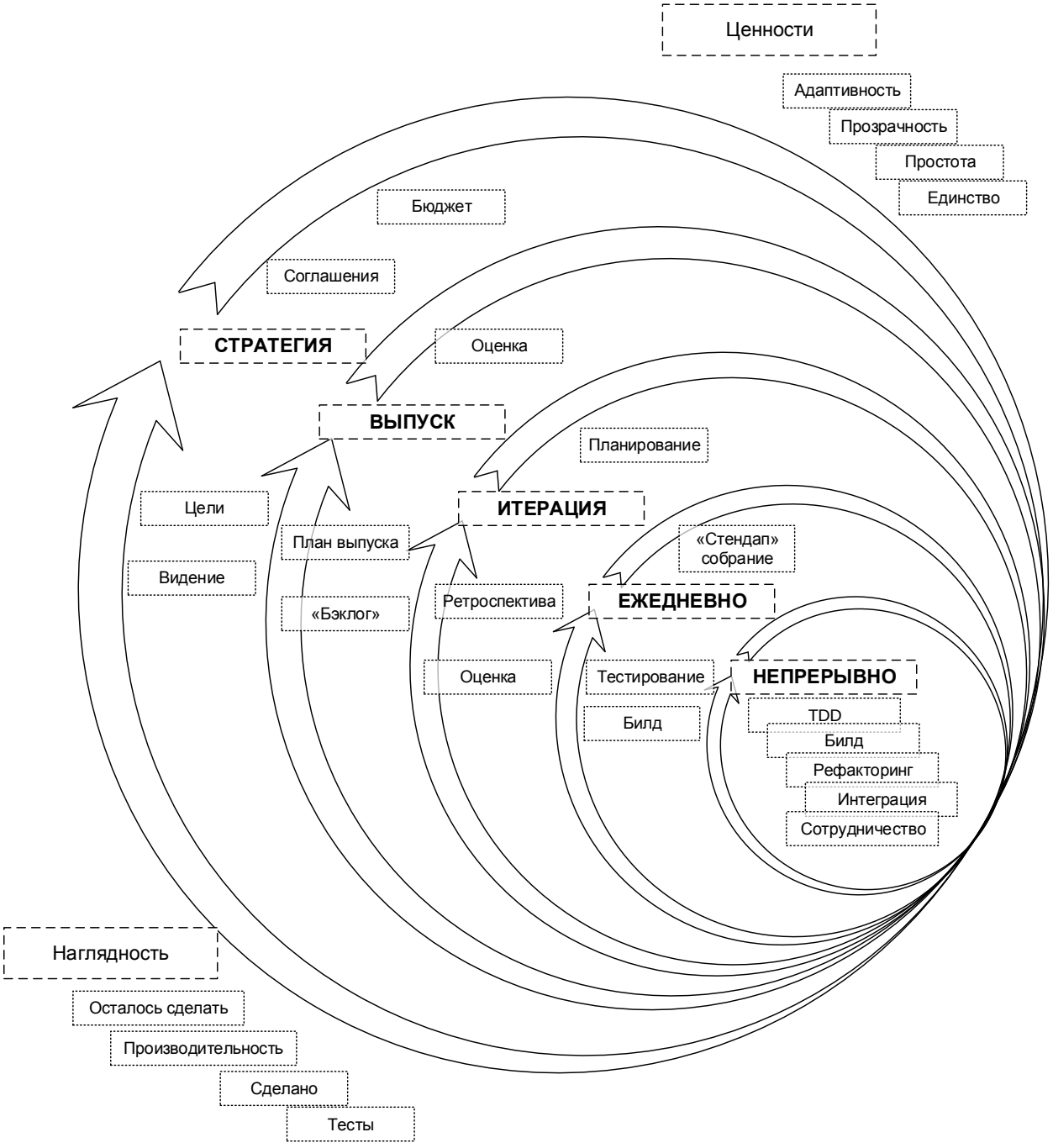

Рисунок 2.1.e — Суть гибкой модели разработки ПО

Очень упрощённо (почти на грани допустимого) можно сказать, что гибкая модель представляет собой облегчённую с точки зрения документации смесь итерационной инкрементальной и спиральной моделей (рисунки 2.1.e[37], 2.1.f); при этом следует помнить об «agile-манифесте» и всех вытекающих из него преимуществах и недостатках.

[37] Оригинал рисунка и исходный материал: http://www.gtagile.com/GT_Agile/Home.html

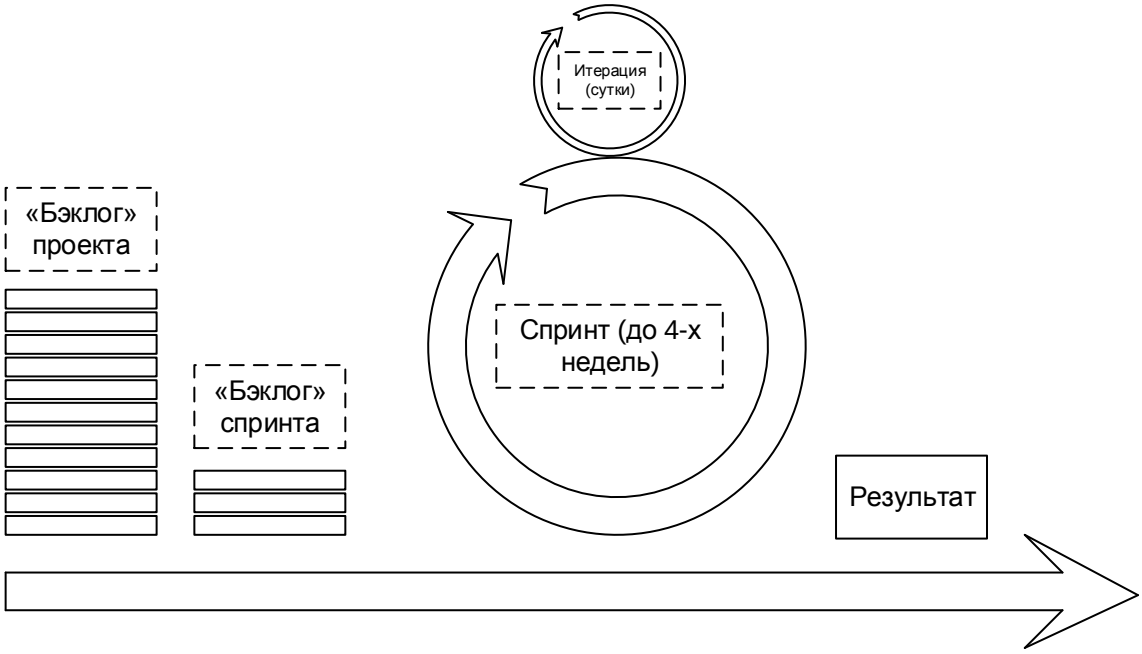

Рисунок 2.1.f — Итерационный подход в рамках гибкой модели и scrum

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

Тем не менее можно утверждать, что всё больше и больше проектов начинают использовать гибкую модель разработки.

Очень подробное и элегантное изложение принципов применения гибкой модели разработки ПО можно найти в статье «The Agile System Development Life Cycle»[38].

Вкратце можно выразить суть моделей разработки ПО таблицей 2.1.a.

Таблица 2.1.a — Сравнение моделей разработки ПО

Модель	Преимущества	Недостатки	Тестирование
Водопадная	• У каждой стадии есть чёткий проверяемый результат. • В каждый момент времени команда выполняет один вид работы. • Хорошо работает для небольших задач.	• Полная неспособность адаптировать проект к изменениям в требованиях. • Крайне позднее создание работающего продукта.	• С середины проекта.
V-образная	• У каждой стадии есть чёткий проверяемый результат. • Внимание тестированию уделяется с первой же стадии. • Хорошо работает для проектов со стабильными требованиями.	• Недостаточная гибкость и адаптируемость. • Отсутствует раннее прототипирование. • Сложность устранения проблем, пропущенных на ранних стадиях развития проекта.	• На переходах между стадиями.

[38] «The Agile System Development Life Cycle» [http://www.ambysoft.com/essays/agileLifecycle.html]

Итерационная инкрементальная	• Достаточно раннее прототипирование. • Простота управления итерациями. • Декомпозиция проекта на управляемые итерации.	• Недостаточная гибкость внутри итераций. • Сложность устранения проблем, пропущенных на ранних стадиях развития проекта.	• В определённые моменты итераций. • Повторное тестирование (после доработки) уже проверенного ранее.
Спиральная	• Глубокий анализ рисков. • Подходит для крупных проектов. • Достаточно раннее прототипирование.	• Высокие накладные расходы. • Сложность применения для небольших проектов. • Высокая зависимость успеха от качества анализа рисков.	
Гибкая	• Максимальное вовлечение заказчика. • Много работы с требованиями. • Тесная интеграция тестирования и разработки. • Минимизация документации.	• Сложность реализации для больших проектов. • Сложность построения стабильных процессов.	• В определённые моменты итераций и **в любой необходимый момент**.

Ещё два кратких и информативных сравнения моделей жизненного цикла ПО можно найти в статьях «Project Lifecycle Models: How They Differ and When to Use Them»[39] и «Блок-схема выбора оптимальной методологии разработки ПО»[40]. А общий обзор всех моделей в контексте тестирования ПО представлен в статье «What are the Software Development Models?»[41].

Задание 2.1.a: представьте, что на собеседовании вас попросили назвать основные модели разработки ПО, перечислить их преимущества и недостатки с точки зрения тестирования. Не ждите собеседования, ответьте на этот вопрос сейчас, а ответ запишите.

[39] «Project Lifecycle Models: How They Differ and When to Use Them» [http://www.business-esolutions.com/islm.htm]
[40] «Блок-схема выбора оптимальной методологии разработки ПО» [http://megamozg.ru/post/23022/]
[41] «What are the Software Development Models?» [http://istqbexamcertification.com/what-are-the-software-development-models/]

2.1.2. Жизненный цикл тестирования

Следуя общей логике итеративности, превалирующей во всех современных моделях разработки ПО, жизненный цикл тестирования также выражается замкнутой последовательностью действий (рисунок 2.1.g).

Важно понимать, что длина такой итерации (и, соответственно, степень подробности каждой стадии) может варьироваться в широчайшем диапазоне — от единиц часов до десятков месяцев. Как правило, если речь идёт о длительном промежутке времени, он разбивается на множество относительно коротких итераций, но сам при этом «тяготеет» к той или иной стадии в каждый момент времени (например, в начале проекта больше планирования, в конце — больше отчётности).

Также ещё раз подчеркнём, что приведённая схема — не догма, и вы легко можете найти альтернативы (например, здесь[42] и здесь[43]), но общая суть и ключевые принципы остаются неизменными. Их и рассмотрим.

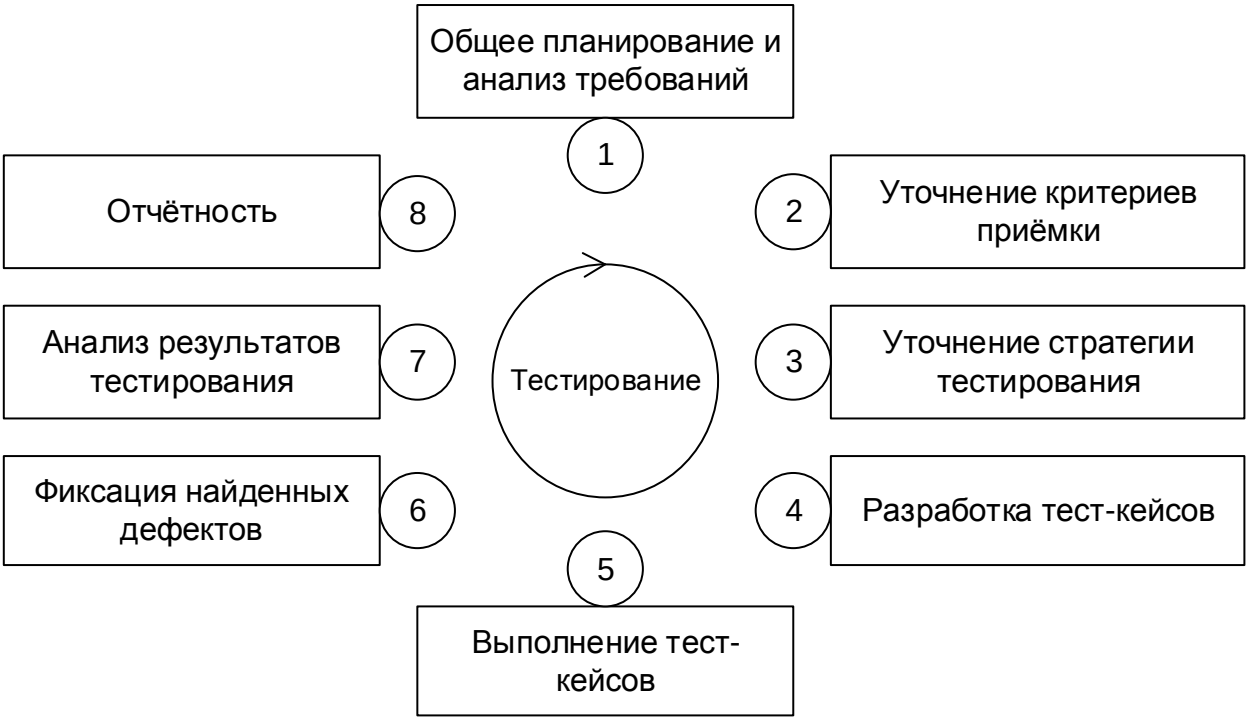

Рисунок 2.1.g — Жизненный цикл тестирования

Стадия 1 (общее планирование и анализ требований) объективно необходима как минимум для того, чтобы иметь ответ на такие вопросы, как: что нам предстоит тестировать; как много будет работы; какие есть сложности; всё ли необходимое у нас есть и т.п. Как правило, получить ответы на эти вопросы невозможно без анализа требований, т.к. именно требования являются первичным источником ответов.

Стадия 2 (уточнение критериев приёмки) позволяет сформулировать или уточнить метрики и признаки возможности или необходимости начала тестирования (entry criteria[44]), приостановки (suspension criteria[45]) и возобновления (resumption

[42] «Software Testing Life Cycle» [http://softwaretestingfundamentals.com/software-testing-life-cycle/]

[43] «Software Testing Life Cycle» [http://www.softwaretestingmentor.com/software-testing-life-cycle/]

[44] **Entry criteria.** The set of generic and specific conditions for permitting a process to go forward with a defined task, e.g. test phase. The purpose of entry criteria is to prevent a task from starting which would entail more (wasted) effort compared to the effort needed to remove the failed entry criteria. [ISTQB Glossary]

[45] **Suspension criteria.** The criteria used to (temporarily) stop all or a portion of the testing activities on the test items. [ISTQB Glossary]

criteria[46]) тестирования, завершения или прекращения тестирования (exit criteria[47]).

Стадия 3 (уточнение стратегии тестирования) представляет собой ещё одно обращение к планированию, но уже на локальном уровне: рассматриваются и уточняются те части стратегии тестирования (test strategy[48]), которые актуальны для текущей итерации.

Стадия 4 (разработка тест-кейсов) посвящена разработке, пересмотру, уточнению, доработке, переработке и прочим действиям с тест-кейсами, наборами тест-кейсов, тестовыми сценариями и иными артефактами, которые будут использоваться при непосредственном выполнении тестирования.

Стадия 5 (выполнение тест-кейсов) и **стадия 6** (фиксация найденных дефектов) тесно связаны между собой и фактически выполняются параллельно: дефекты фиксируются сразу по факту их обнаружения в процессе выполнения тест-кейсов. Однако зачастую после выполнения всех тест-кейсов и написания всех отчётов о найденных дефектах проводится явно выделенная стадия уточнения, на которой все отчёты о дефектах рассматриваются повторно с целью формирования единого понимания проблемы и уточнения таких характеристик дефекта, как важность и срочность.

Стадия 7 (анализ результатов тестирования) и **стадия 8** (отчётность) также тесно связаны между собой и выполняются практически параллельно. Формулируемые на стадии анализа результатов выводы напрямую зависят от плана тестирования, критериев приёмки и уточнённой стратегии, полученных на стадиях 1, 2 и 3. Полученные выводы оформляются на стадии 8 и служат основой для стадий 1, 2 и 3 следующей итерации тестирования. Таким образом, цикл замыкается.

В жизненном цикле тестирования пять из восьми стадий так или иначе связаны с управлением проектами, рассмотрение которого не входит в наши планы, а потому обо всём, что касается планирования и отчётности, мы кратко поговорим в главе «Оценка трудозатрат, планирование и отчётность»[{203}]. А сейчас мы переходим к ключевым навыкам и основным видам деятельности тестировщиков и начнём с работы с документацией.

[46] **Resumption criteria.** The criteria used to restart all or a portion of the testing activities that were suspended previously. [ISTQB Glossary]

[47] **Exit criteria.** The set of generic and specific conditions, agreed upon with the stakeholders for permitting a process to be officially completed. The purpose of exit criteria is to prevent a task from being considered completed when there are still outstanding parts of the task which have not been finished. Exit criteria are used to report against and to plan when to stop testing. [ISTQB Glossary]

[48] **Test strategy.** A high-level description of the test levels to be performed and the testing within those levels for an organization or program (one or more projects). [ISTQB Glossary]