**Тестирование в модели жизненного цикла разработки ПО**

Под моделью ЖЦ понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач, выполняемых на протяжении ЖЦ. Модель ЖЦ зависит от специфики ИС и специфики условий, в которых последняя создается и функционирует. Стандарт ISO/IEC 12207 не предлагает конкретную модель ЖЦ и метод разработки ПО. Его регламенты являются общими для любых моделей ЖЦ, методологий и технологий разработки. Стандарт описывает структуру процессов ЖЦ ПО, но не конкретизирует в деталях, как реализовать или выполнить действия и задачи, включенные в эти процессы.

С точки зрения ISO 9126, Качество (программных средств) можно определить как совокупную характеристику исследуемого ПО с учётом следующих составляющих:

* надёжность;
* сопровождаемость;
* практичность;
* эффективность;
* мобильность;
* функциональность;

Более полный список атрибутов и критериев можно найти в стандарте ISO 9126 Международной организации по стандартизации. Состав и содержание документации, сопутствующей процессу тестирования, определяется стандартом IEEE 829-1998 Standard for Software Test Documentation.

К настоящему времени наибольшее распространение получили следующие основные модели ЖЦ:

* [Каскадная модель](http://www.4stud.info/software-construction-and-testing/lecture6.html) (70-85 г.г.), так же известная как *водопадная* или *последовательная*;
* [Итеративная и инкрементальная](http://www.4stud.info/software-construction-and-testing/lecture8.html) – эволюционная (гибридная, смешанная) модель (86-90 г.г.);
* *Спиральная* (spiral) модель или модель Боэма (88-90-е г.г.).

Легко обнаружить, что в разное время и в разных источниках приводится разный список моделей и их интерпретация. Например, ранее, инкрементальная модель понималась как построение системы в виде последовательности сборок (релизов), определенной в соответствии с заранее подготовленным планом и заданными (уже сформулированными) и неизменными требованиями. Сегодня об инкрементальном подходе чаще всего говорят в контексте постепенного наращивания функциональности создаваемого продукта.

Может показаться, что индустрия пришла, наконец, к общей “правильной” модели. Однако, каскадная модель, многократно “убитая” и теорией и практикой, продолжает встречаться в реальной жизни. Спиральная модель является ярким представителем эволюционного взгляда, но, в то же время, представляет собой единственную модель, которая уделяет явное внимание анализу и предупреждению рисков. Коротко рассмотрим каждую из моделей жизненного цикла.

В первую очередь нужно отметить, что вопросы тестирования следует рассматривать в контексте всего жизненного цикла ПО, начиная от разработки ТЗ и заканчивая сопровождением приложений. Как известно, тестирование — это процедура обнаружения дефектов (ошибок) ПО до его промышленного использования. Очевидно, что трудоемкость такой работы связана с количеством самих ошибок, в связи с чем надо четко выделить основные причины их появления:

* неудовлетворительное организационное, методическое и техническое обеспечение всего процесса разработки;
* сжатые сроки исполнения проекта;
* сложность проекта, большое число требований и их изменений по ходу работы;
* недостаточная квалификация разработчиков.

Тестирование является лишь составляющей частью отладки — процесса доводки ПО после его написания до эксплуатационного состояния. Процесс этот включает две основные процедуры: обнаружение ошибок (тестирование) и поиск и устранение их причин. Однако, даже учитывая все возможные взаимосвязи этих работ (например, поиск причин ошибок требует проведения специального дополнительного тестирования), нужно подчеркнуть, что тестирование является достаточно автономным, независимым этапом жизненного цикла ПО.

Тестирование пронизывает весь жизненный цикл ПО, начиная от проектирования и заканчивая неопределенно долгим этапом эксплуатации. Эти работы напрямую связаны с задачами управления требованиями и изменениями, ведь целью тестирования является как раз возможность убедиться в соответствии программ заявленным требованиям.

Тестирование — процесс пошаговый. Наверное, имеет смысл разделить проверку работоспособности программ в ходе непосредственного написания кода (самим программистом) и после завершения основного этапа кодирования (скорее всего, специальными тестировщиками). Тут можно вспомнить о золотом правиле программирования: написание каждых 20—30 строк кода (тем более законченных процедур, функций) должно сопровождаться проверкой их работоспособности, хотя бы в каком-то основном режиме. В то же время нужно подчеркнуть и важное различие в проведении тестирования в ходе кодирования и по его завершении: в первом случае продолжать написание программы (а также запуск других тестовых примеров) желательно только после устранения ошибки, во втором осуществляется пакетное выполнение серии текстов с простой фиксацией их результатов.

Тестирование — процесс также итерационный. После обнаружения и исправления каждой ошибки обязательно следует повторение тестов, чтобы убедиться в работоспособности программы. Более того, для идентификации причины обнаруженной проблемы может потребоваться проведение специального дополнительного тестирования. При этом нужно всегда помнить о фундаментальном выводе Дейкстры, вынесенном в эпиграф этой лекции.

Модель тестирования ПО на основе обобщенной модели ЖЦ ПО

Обобщенная модель жизненного цикла тестирования ПО может быть представлена в виде буквы V, как показано на рис. 4. Такая конценпция получила соответствующее название - V-модель.

Это один из способов продемонстрировать, как соотносятся процессы тестирования с основными процессами проектирования и разработки. В V-модели основные этапы ЖЦ ПО образуют левую сторону “V”, кодирование находится в самой нижней точке диаграммы, а тестирование образует ее правую сторону. Для простоты изложения, вид деятельности, подобный сопровождению, на диаграмме не показан.

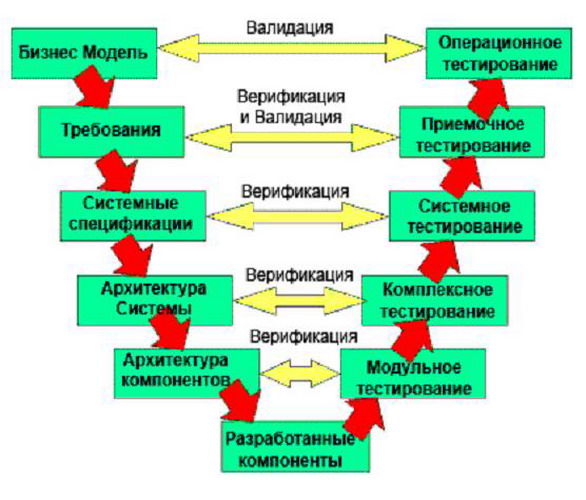


Рис. 4. V-образная модель обобщенного ЖЦ ТП

Следует уточнить, что, точно так же как обобщенная модель ЖЦ ПО не определяет строгой последовательности этапов разработки и не исключает цикличность и пересмотр всех или некоторых из них, так и “V”-модель – лишь показывает основные этапы тестирования ПО и их связь с этапами разработки. Обобщенная модель ЖЦ тестирования ПО может быть преобразована к виду конкретной модели ЖЦ ПО. Так, при работе по итеративной или спиральной модели, “V”-модель тестирования приобретает итеративную природу, как и другие процессы разработки. Это означает, что обозначенные виды (уровни) тестирования могут неоднократно повторятся в процессе ЖЦ ПО в каждой итерации разработки, а также – в качестве регрессионного тестирования уже разработанных и протестированных на предыдущих итерациях элементов системы.

Таким образом, на основе современных методологий, стандартов качества и стандартов разработки ПО мы систематизировали общее понимание тестирования и определили обобщенную модель ЖЦ ТП, которая обладает следующими качествами:

ЖЦ ТП систематически связана с ЖЦ ПО;

ЖЦ ТП закрывает все уровни проверки качества ПО;

ЖЦ ТП используется для валидации и верификации, которые являются крайне важными процессами по обеспечению качества ПО; даже название обобщенной модели ЖЦ ТП “V-модель” отражает свой главный смысл – Верификация и Валидация (Verification and Validation) качества ПО, как неотъемлемая составляющая любой современной разработки;

ЖЦ ТП позволяет понять, что, как и для чего должно быть подвержено тестированию в каждой точке ЖЦ ПО;

Обобщенная модель ЖЦ ТП может быть преобразована под любую модель ЖЦ ПО. В частности, она может быть преобразована под методологию RUP, приобретая цикличность в двух направлениях – горизонтальном (верификация на каждом уровне) и вертикальном (валидация по всем уровням).

Таким образом, тестирование вправе называться самостоятельной дисциплиной, которая имеет свои цели, задачи, роли, виды, стратегии, методы, критерии, свою методологию и технологию

Классификация тестирования на уровни, виды и типы

Тесты существенно различаются по задачам, которые с их помощью решаются, и по используемой технике. Различие задач тестирования приводит, естественным образом, к необходимости использовать весьма разнообразные типы (виды) тестирования. Принято подразделять тестирование на виды по следующим категориям:

по объектам (элементам) тестирования, часто разделение на виды тестов по данному критерию называют разделением тестирования на уровни;

по глубине тестирования, то есть разделение тестовых испытаний на типы проводится в зависимости от количества времени и объема тестируемых компонент программного продукта.

Тем не менее, основная классификация тестов на виды производится в соответствие с традиционными показателями качества, которые проверяются с их помощью.

12.1. Уровни тестирования

Модульное тестирование (Автономное или Unit-тестирование)

Входные требования — Архитектура компонентов или модель “нижнего уровня” системы (Component Design или Low Level Design)

Объект тестирования — Разработанные компоненты

Определение: На данном уровне тестируются по отдельности небольшие элементы системы, максимально отделенные от других элементов и, в то же время, пригодные для тестирования. Такое тестирование обычно проводится сразу же вслед за разработкой каждого из элементов и направлено на оценку соответствия функциональности каждого из компонентов спроектированной “модели компонентов”.

Комплексное тестирование (Сборочное тестирование, integration testing или interface testing)

Входные требования — Архитектура системы или модель “верхнего уровня” системы (System Design или High Level Design)

Объект тестирования — Собранная из компонентов система или подсистема

Определение: На данном уровне тестируются объединенные элементы (компоненты или подсистемы) общей системы, чаще всего некоторая взаимодействующая между собой группа элементов.

Комплексное тестирование направлено не на проверку функционирования каждого из компонентов, а на проверку взаимодействия компонентов в соответствии с «Архитектурой системы».

Тесты данного уровня обычно проверяют все интерфейсы взаимодействия между компонентами, определенные в системной архитектуре, до тех пор, пока все компоненты не будут разработаны, отлажены и проинтегрированы друг с другом в единую систему.

Системное тестирование (system testing)

Входные требования — Системные спецификации (System Specification)

Объект тестирования — Разработанная система

Определение: После того, как система собрана из составляющих компонентов, она должна быть протестирована на соответствие “Системным спецификациям” – реализованы ли все функциональные и нефункциональные требования к разрабатываемой системе.

На данном уровне тестируется приложение или система (одно или более приложений) целиком.

Приемочное тестирование (Приемо-сдаточное тестирование или acceptance testing)

Входные требования — Требования (Requirements)

Объект тестирования — Разработанная система

Определение: На данном уровне завершенное приложение (система) тестируется Заказчиком, конечными пользователями или соответствующими уполномоченными с целью определения соответствия системы “Требованиям Заказчика” и готовности системы к внедрению. Приемосдаточные испытания оформляют процесс передачи продукта от Разработчика Заказчику. В зависимости от особенностей продукта и от требований Заказчика они могут проводиться в различной форме. Например, в виде альфа- или бета-тестирования.

Приемочное тестирование схоже с системным тестированием, но со следующим различием:

Системное тестирование проверяет, что разработанная система соответствует специфицированным требованиям;

Приемочное тестирование проверяет, что разработанная система удовлетворяет запрошенным Заказчиком требованиям с упором на нужды конечных пользователей в данной предметной области.

Операционное тестирование (Release Testing)

Входные требования — Бизнес модель (Business Case или Business Model)

Объект тестирования — Разработанная система

Определение: Даже если система удовлетворяет всем требованиям, важно убедиться в том, что она удовлетворяет нуждам пользователя и выполняет свою роль в среде своей эксплуатации, как это было определено в бизнес-модели системы. Следует учесть, что и Бизнес модель может содержать ошибки. Поэтому так важно провести операционное тестирование как финальный шаг Валидации.

Кроме этого, тестирование в среде эксплуатации позволяет выявить и нефункциональные проблемы, такие как: конфликт с другими системами, смежными в области бизнеса или в программных и электронных окружениях; недостаточная производительность системы в среде эксплуатации и др.

Очевидно, что нахождение подобных вещей на стадии внедрения – критичная и дорогостоящая проблема. Поэтому так важно проведение не только верификации, но и валидации, с самых ранних этапов разработки ПО.

Основное разделение тестов на виды по объектам тестирования, или, точнее, на уровни тестирования, было произведено нами при определении обобщенной модели ЖЦ ТП. Уровни тестирования приведены ниже. Для каждого уровня тестирования могут использоваться различные виды тестирования, для каждого из которых, в свою очередь, могут использоваться различные типы тестовых испытаний.

12.2. Виды тестирования

Инсталляционное тестирование (Installation testing)

Определение: В процессе инсталляционного тестирования проверяется корректность установки и деинсталляции программного продукта в среде максимально приближенной к эксплуатационной. Проверка правильности установки программного продукта должна быть обязательным элементом проекта по тестированию любого продукта.

Цель: Основная цель состоит в том, чтобы убедиться, что продукт может быть установлен/деинсталлирован при различных условиях – таких как: новая инсталляция, усовершенствование системы (upgrade), установка по умолчанию, полная установка, установка по выбору.

Дымное тестирование (проверка на дым, Smoke testing)

Определение: Первый прогон программы (после написания или после внесения существенных изменений). Как правило, используется для определения, готова ли программа для проведения более обширного тестирования.

Цель: Выявление проблем «лежащих на поверхности» – тестируется чаще всего основная бизнес логика программы

Функциональное тестирование (Functional testing)

Определение: Проверка соответствия продукта функциональным требованиям и спецификациям

Цель: Проверка соответствия продукта функциональным требованиям и спецификациям

[Регрессионное тестирование](http://www.4stud.info/software-construction-and-testing/regression-testing.html) (Regression testing)

Определение: Повторное тестирование после внесения изменений в программное обеспечение или в его окружение (в новой версии приложения), чтобы убедиться в том, что функции, которые работали в предыдущей версии системы, по-прежнему работают так, как ожидалось, а найденные дефекты успешно исправлены (все протестированное ранее тестируется повторно)

Цель: Выявление потенциальных проблем, которые могли возникнуть в результате изменений. Проверка исправления найденных ранее дефектов.

Интеграционное тестирование (Integration testing)

Определение: Проверка скомбинированных компонентов прикладной программы с целью определения корректности их совместного функционирования

Цель: Выявление потенциальных проблем в совместном функционировании компонент

Тестирование графического интерфейса пользователя (User Interface testing)

Определение: Тестирование интерфейса – экранов, кнопок и т.д. Большая часть функциональности ПО реализуется, как правило, через пользовательский интерфейс.

Цель: Обнаружение ошибок в интерфейсе и поиск ошибок в функциональности посредством интерфейса

Тестирование производительности (Performance testing)

Определение: Проверка скорости работы системы (время отклика, частота транзакций и другие зависящие от времени) в имитационной и реальной средах

Цель: Установить реальную производительность программного продукта

Нагрузочное тестирование (Load testing)

Определение: Это те же тесты производительности, при которых система подвергается различным нагрузкам; при этом цель этого тестирования – оценить способность системы правильно функционировать при некотором превышении планируемых нагрузок при реальной эксплуатации (система имеет некоторый «запас прочности»)

Цель: Убедиться в том, что система работает соответственно ожидаемым рабочим нагрузочным параметрам (какой предел работоспособности)

Стресс тестирование (Stress testing)

Определение: Является одним из разновидностей тестирования на производительность. Проверяется поведение системы при недостатке ресурсов (дискового пространства, обрывов сети и т.д.).

Цель: Проверка того, что система адекватно реагирует на те или иные стрессовые ситуации

Конфигурационное тестирование (Configuration testing)

Определение: Конфигурационное тестирование – тестирование работы на различных платформах. Различные варианты аппаратной конфигурации, версии операционной системы и окружения.

Цель: Проверить работоспособность системы при различных конфигурациях

Тестирование интернационализации (Internationalization testing)

Определение: Этот вид тестирует насколько продукт готов к тому, чтобы быть адаптированном для работы в других локалях с другим языком пользовательского интерфейса, отличном от языка по умолчанию (как правило, это английский)

Цель: Проверить способность продукта быть быстро локализованным под необходимую локаль потенциальных пользователей системы

Локализационное тестирование (Localization testing)

Определение: Локализационное тестирование, в свою очередь, проверяет, правильно ли локализован продукт. То есть, переведен на другой язык и корректно работает с учетом национальных особенностией страны или региона, в котором будет продаваться и использоваться продукт.

Цель: Проверить, правильно ли локализован продукт

Классификация тестов на виды производится в соответствие с традиционными показателями качества, которые проверяются с их помощью. Иными словами, разделение тестирования на виды происходит в зависимости от типа требований (функциональные, нефункциональные), проверяемых с помощью тестов.

Для проверки функциональности (functionality) ПО необходимо испытать приложенние на выполнение функциональных требований к нему (сценариев использования и др.). Для этого используются собственно функциональные тесты, а также тесты безопасности, объема и другие.

Тестирование надежности (reliability) ПО производится с целью проверки нефункциональных требований, что приложение работает, как и ожидалось, устойчиво к падениям и т.п. Здесь применяются интеграционные тесты, тесты структуры, стрессовые тесты и другие.

Тестирование удобства использования (usability) ПО (нефункциональные требования) производится с целью удостовериться в том, что приложение удобно для использования его конечным пользователям. Включает в себя тесты на человеческий фактор, эстетику интерфейса и его непротиворечивость, наличие и качество оперативной и контекстной помощи, руководств и учебных материалов.

Тестирование производительности (performance) ПО выполняется с целью удостовериться, что функционирование приложения обеспечивается в то время, когда выполняются нефункциональные требования к приложению по работе в реальных условиях. Включает в себя оценку временных профилей, времени отклика, операционной надежности и некоторых других характеристик. Основные виды тестирования приведены ниже.

12.3. Типы тестовых испытаний по глубине тестирования.

Приемочный тест (Smoke test) – первый и самый короткий тест, призванный проводить проверку основных элементов программного продукта и его работоспособности в целом. В случае функционального тестирования – проверяется основной функционал приложения. Тест занимает 1-4 часа в зависимости от сложности тестируемого продукта. На основе результатов данного теста принимается решение о приемке версии программного продукта и продолжении тестирования текущей версии продукта более серьезными тестовыми испытаниями.

Критический тест (Critical path test) – основной тип тестовых испытаний, во время которого значимые элементы и функции приложения проверяются на предмет правильности работы при стандартном их использовании. Как правило, на данном уровне тестирования проверяется основная масса требований к продукту.

Расширенный тест (Extended test) – вид углубленного тестирования, при котором проверяется нестандартное использование программного продукта, границы переполнения массивов данных, ввод специальных символов и т.п.