**SDLC и STLC**

**SDLC** - это систематизированный процесс, этапы которого охватывают полный жизненный цикл программного обеспечения (Software Lifecycle) и который определяет различные этапы разработки программного обеспечения для создания высококачественного программного обеспечения, отвечающего ожиданиям клиентов и для улучшения эффективности разработки. Разработка системы должна быть завершена в заранее определенные сроки и стоимость. Каждая фаза жизненного цикла SDLC имеет свой собственный процесс и результаты, которые используются в следующей фазе.

**Фазы SDLC**:

* **Сбор и анализ требований** (Requirement Gathering and Analysis): На этом этапе от клиента собирается вся необходимая информация для разработки продукта в соответствии с их ожиданиями. Любые неясности должны быть разрешены сразу на этом этапе. Бизнес-аналитик и менеджер проекта назначили встречу с заказчиком, чтобы собрать всю информацию, например, что заказчик хочет построить, кто будет конечным пользователем, какова цель продукта. Перед созданием продукта очень важно понимание или знание продукта. Например, клиент хочет иметь приложение, которое включает денежные транзакции. В этом случае требование должно быть четким, например, какие транзакции будут выполняться, как они будут проводиться, в какой валюте они будут проводиться и т. д. После того, как сбор требований завершен, проводится анализ для проверки возможности разработки продукта. После четкого понимания требования создается документ SRS (Спецификация требований к программному обеспечению). Этот документ должен быть полностью понят разработчикам, а также должен быть рассмотрен заказчиком для использования в будущем;
* **Дизайн** (Design): На этом этапе требования, собранные в документе SRS, используются в качестве входных данных, и создается архитектура программного обеспечения, которая используется для реализации разработки системы. Создаются два вида дизайн-документов:
  + Высокоуровневый дизайн (HLD - High-Level Design):
    - Краткое описание и название каждого модуля;
    - Краткое описание функциональности каждого модуля;
    - Отношения интерфейсов и зависимости между модулями;
    - Таблицы базы данных, идентифицированные вместе с их ключевыми элементами;
    - Полные архитектурные схемы с подробными сведениями о технологиях.
  + Низкоуровневый дизайн (LLD - Low-Level Design):
    - Функциональная логика модулей;
    - Таблицы базы данных, которые включают тип и размер;
    - Полная детализация интерфейсов;
    - Решение всех типов проблем с зависимостями;
    - Список сообщений об ошибках;
    - Полные входные и выходные значения для каждого модуля.
* **Разработка** (Implementation or Coding): Реализация / кодирование начинается, как только разработчик получает Design document. Дизайн программного обеспечения переведен в исходный код. На этом этапе реализуются все компоненты программного обеспечения;
* **Тестирование** (Testing): Тестирование начинается после завершения кодирования и выпуска модулей для тестирования. На этом этапе разработанное программное обеспечение тщательно тестируется, и все обнаруженные дефекты передаются разработчикам для их исправления. Повторное тестирование, регрессионное тестирование проводится до тех пор, пока программное обеспечение не будет соответствовать ожиданиям клиента. Тестировщики обращаются к документу SRS, чтобы убедиться, что программное обеспечение соответствует стандарту заказчика;
* **Развертывание** (Deployment): После тестирования продукта он развертывается в производственной среде или выполняется первое UAT (пользовательское приемочное тестирование), в зависимости от ожиданий клиента. В случае UAT создается копия производственной среды, и заказчик вместе с разработчиками выполняет тестирование. Если клиент остается доволен, то предоставляет согласие на релиз;
* **Поддержка** (Maintenance): Основное внимание на этом этапе SDLC уделяется обеспечению того, чтобы потребности продолжали удовлетворяться и чтобы система продолжала работать в соответствии со спецификацией, упомянутой в первом этапе. После того, как система развернута и клиенты начинают использовать разработанную систему следует 3 вида активностей:
  + Исправление ошибок;
  + Обновление;
  + Улучшение.

**STLC** - это процесс тестирования, который включает в себя определенную последовательность шагов, чтобы гарантировать достижение целей в области качества. В процессе STLC каждое действие выполняется планомерно и систематически. Каждый этап имеет разные цели и результаты. У разных организаций разные этапы STLC, однако, основа остается прежней.

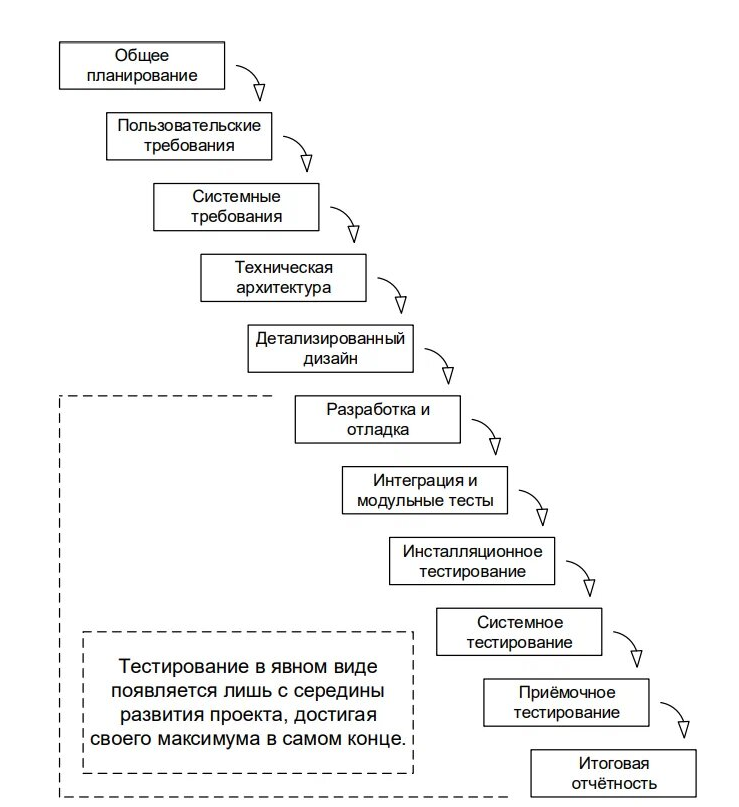
STLC имеет несколько взаимосвязанных фаз и в целом очень похож на SDLC. Эти фазы являются последовательными и называются:

* **Анализ требований** (Requirement Analysis): один из важнейших этапов, потому что именно на нем можно почти бесплатно исправить недостатки проекта. Этап анализа требований также определяет потенциальную потребность в автоматизированном тестировании и позволяет производить экономические расчеты затрат на рабочую силу на основе оценки проекта. На этом же этапе обсуждаются и документируются критерии начала и окончания тестирования.
  + Entry Criteria: BRS (Business Requirement Specification)
  + Deliverables: список всех проверяемых требований, технико-экономическое обоснование автоматизации (если применимо);
* **Планирование тестирования** (Test Planning): на этом этапе формируется план тестирования, т.е. мы определяем действия и ресурсы, которые помогут достичь целей тестирования (участники и их роли, инструменты, окружение). Во время планирования мы также пытаемся определить метрики, метод сбора и отслеживания этих метрик. План составляют исходя из требований, тестовой стратегии и анализа рисков.
  + Entry Criteria: Requirements Documents;
  + Deliverables: Test Strategy, Test Plan, and Test Effort estimation document.
* **Разработка тест-кейсов** (Test Case Development): подразумевает использование ручного и автоматизированного тестирования для достижения полного охвата функциональности программного обеспечения, при этом процесс основан на заранее установленных требованиях. Чаще всего тест-кейсы для автоматического тестирования пишутся отдельно, так как кейсы для ручного тестирования описаны в виде шпаргалок (cheat sheets).
  + Entry Criteria: Requirements Documents (Updated version);
  + Deliverables: Test cases, Test Scripts (if automation), Test data.
* **Настройка тестовой среды** (Test Environment Setup): в плане тестирования четко указано, какую тестовую среду следует использовать. На этом этапе STLC настраиваются операционные системы и виртуальные машины, развертываются инструменты тестирования, такие как Selenium, Katalon Studio, а также тестовая среда и базы данных проекта. Мы также обращаемся с запросами к DevOps и администраторам, если требуется поддержка.
  + Entry Criteria: Test Plan, Smoke Test cases, Test Data;
  + Deliverables: Test Environment. Smoke Test Results.
* **Выполнение тестов** (Test Execution): тесты выполняются на основе готовой тестовой документации и правильно настроенной тестовой среды. Все результаты тестирования регистрируются в Системе управления тестированием. Отрицательно пройденные тесты, в которых фактический результат отличается от ожидаемого, регистрируются как ошибки и передаются команде разработчиков на доработку с последующей перепроверкой после исправления.
  + Entry Criteria: Test Plan document, Test cases, Test data, Test Environment;
  + Deliverables: Test case execution report, Defect report, RTM.
* **Завершение цикла испытаний** (Test Cycle Closure): окончательная генерация отчетов о тестировании для клиента. Они должны включать затраченное время, процент обнаруженных ошибок и положительных результатов тестирования, общее количество обнаруженных и исправленных ошибок. Что касается отдела тестирования, то это момент для анализа его работы, подведения итогов, анализа его продуктивности и возможности внести предложения по улучшению качества тестирования.
  + Entry Criteria: Test Case Execution report (убедитесь, что нет открытых high severity defects), Defect report;
  + Deliverables: Test Closure report, Test metrics.

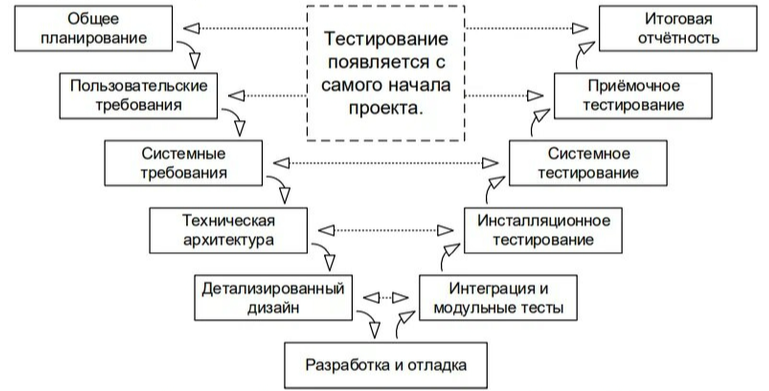


**Модель разработки ПО**

**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.

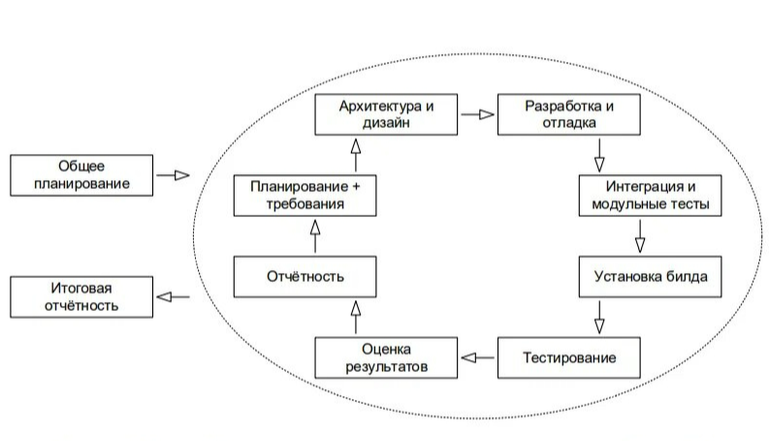
**Водопадная модель** (waterfall model) - сейчас представляет скорее исторический интерес, т.к. в современных проектах практически неприменима, исключая авиастроение, военную или космическую отрасли, медицину и финансовый сектор. Она предполагает однократное выполнение каждой из фаз проекта, которые, в свою очередь, строго следуют друг за другом. Очень упрощенно 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. 

**V-модель (V-model) -** Модель, описывающая процессы жизненного цикла разработки программного обеспечения с момента составление спецификации требований до этапа сопровождения. V модель показывает интеграцию процессов тестирования в каждую фазу цикла разработки программного обеспечения. (ISTQB)



**Инкрементная модель разработки (incremental development model)** - Модель жизненного цикла разработки, в которой проект разделен на серию приращений, каждое из которых добавляет часть функциональности в общих требованиях проекта. Требования приоритезированы и внедряются в порядке приоритетов. В некоторых (но не во всех) версиях этой модели жизненного цикла каждый подпроект следует «мини V-модели» со своими собственными фазами проектирования, кодирования и тестирования. (ISTQB).

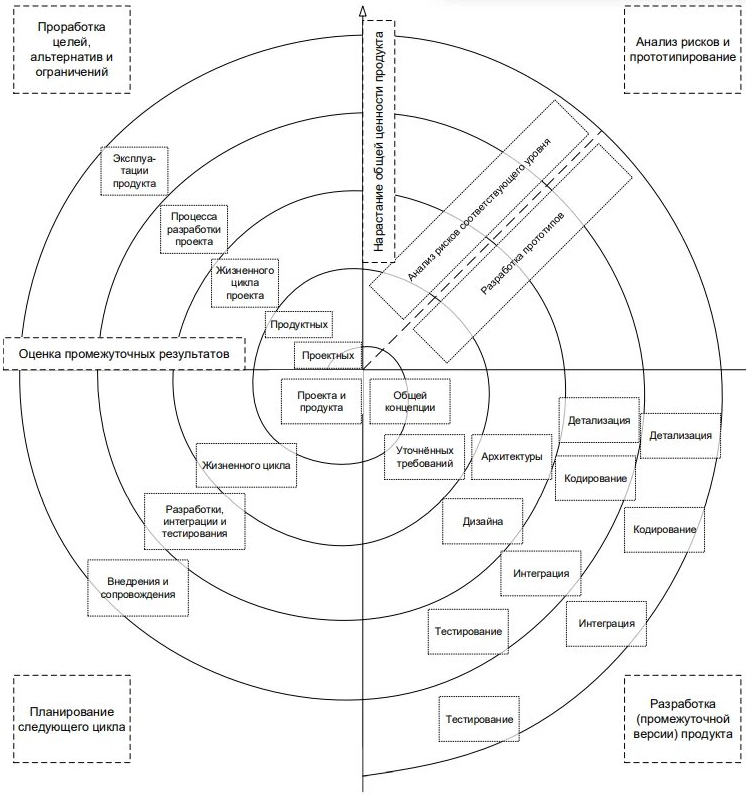
Итерационная инкрементальная модель является фундаментальной основой современного подхода к разработке ПО. Ключевой особенностью данной модели является разбиение проекта на относительно небольшие промежутки (итерации), каждый из которых в общем случае может включать в себя все классические стадии, присущие водопадной и v-образной моделям. Итогом итерации является приращение (инкремент) функциональности продукта, выраженное в промежуточном билде (build).

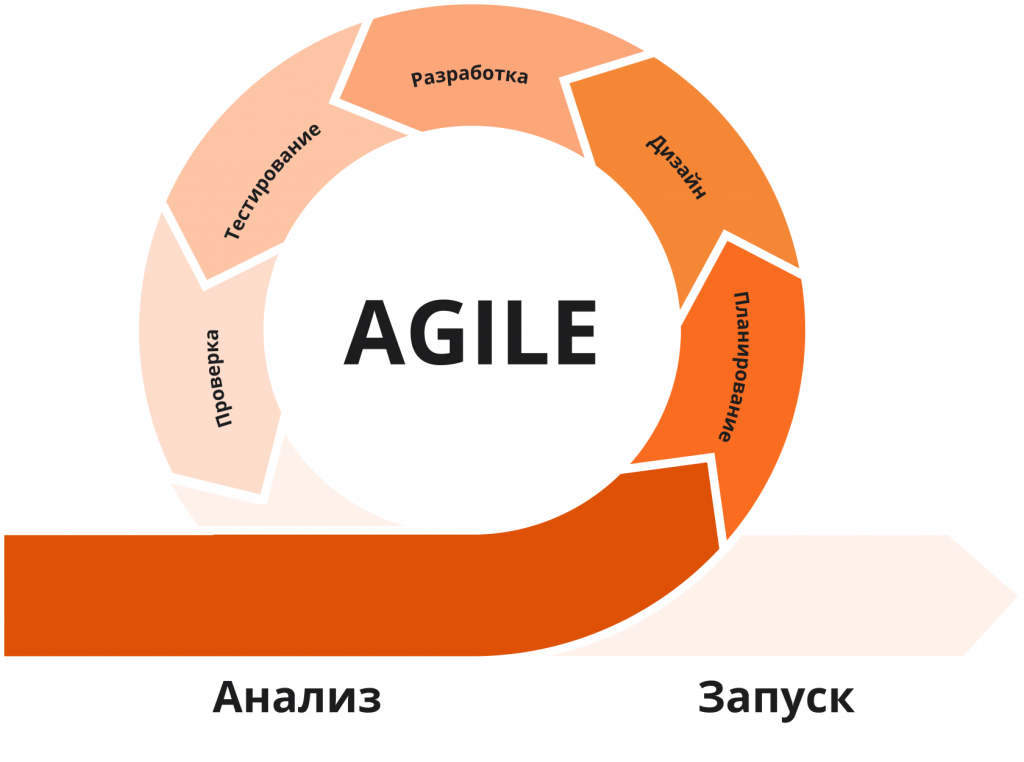


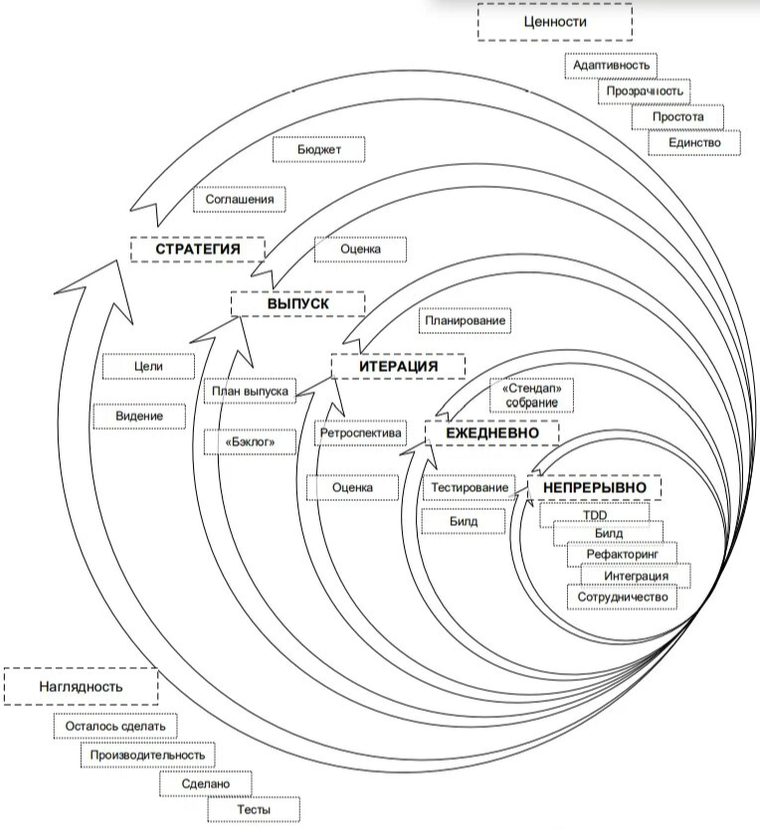
**Спиральная модель** представляет собой частный случай итерационной инкрементальной модели, в котором особое внимание уделяется управлению рисками, в особенности влияющими на организацию процесса разработки проекта и контрольные точки.

Присутствуют четыре ключевые фазы:

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



**Гибкая методология разработки ПО (agile software development)** - Группа методологий разработки программного обеспечения, основанных на итеративной поэтапной разработке, где требования и решения развиваются посредством сотрудничества между самоорганизующимися межфункциональными командами. (ISTQB)



**Подходы к разработке/тестированию**

* **TDD - Test Driven Development**: разработка на основе тестов основывается на повторении коротких циклов разработки: изначально пишется тест, покрывающий желаемое изменение, затем пишется программный код, который реализует желаемое поведение системы и позволит пройти написанный тест. Затем проводится рефакторинг написанного кода с постоянной проверкой прохождения тестов. Есть два уровня TDD:
* **Acceptance TDD (ATDD):** вы пишете один приемочный тест. Этот тест удовлетворяет требованиям спецификации или удовлетворяет поведению системы. После этого пишете достаточно производственного / функционального кода, чтобы выполнить этот приемочный тест. Приемочный тест фокусируется на общем поведении системы. ATDD также известен как BDD - Behavior Driven Development;
* **Developer TDD:** вы пишете один тест разработчика, то есть модульный тест, а затем просто достаточно производственного кода для выполнения этого теста. Модульное тестирование фокусируется на каждой небольшой функциональности системы. Это называется просто TDD. Основная цель ATDD и TDD - определить подробные, выполнимые требования для вашего решения точно в срок (JIT). JIT означает принятие во внимание только тех требований, которые необходимы в системе, что повышает эффективность.
* **BDD - Behaviour Driven Development:** это разработка, основанная на описании поведения. Определенный человек(или люди) пишет описания вида "я как пользователь хочу когда нажали кнопку пуск тогда показывалось меню как на картинке" (там есть специально выделенные ключевые слова). Программисты давно написали специальные тулы, которые подобные описания переводят в тесты (иногда совсем прозрачно для программиста). А дальше классическая разработка с тестами (TDD);
* **TDD - Type Driven Development:** при разработке на основе типов ваши типы данных и сигнатуры типов являются спецификацией программы. Типы также служат формой документации, которая гарантированно обновляется. Типы представляют из себя небольшие контрольные точки, благодаря которым, мы получаем множество мини-тестов по всему нашему приложению;
* **DDD - Domain Driven Design:** Предметно-ориентированное проектирование не является какой-либо конкретной технологией или методологией. DDD - это набор правил, которые позволяют принимать правильные проектные решения. Это набор принципов и схем, направленных на создание оптимальных систем объектов. Процесс разработки сводится к созданию программных абстракций, которые называются моделями предметных областей. В эти модели входит бизнес-логика, устанавливающая связь между реальными условиями области применения продукта и кодом;
* **FDD - Features Driven Development:** представляет собой попытку объединить наиболее признанные в индустрии разработки программного обеспечения методики, принимающие за основу важную для заказчика функциональность (свойства) разрабатываемого программного обеспечения. Основной целью данной методологии является разработка реального, работающего программного обеспечения систематически, в установленные сроки;
* **MDD - Model Driven Development:** разработка, управляемая моделями - это стиль разработки программного обеспечения, когда модели становятся основными артефактами разработки, из которых генерируется код и другие артефакты;
* **PDD - Panic Driven Development:** это своеобразный антипаттерн разработки, который, к сожалению, мы все время от времени практикуем. По сути это то, что получается, когда процессы плохо налажены и команда импровизирует в условиях горящих сроков (новые задачи приоритетнее старых, код решает конкретные срочные задачи, но копится технический долг, тестирование в конце и т.д.);
* **ADD - API Driven Development:** разработка на основе API - это практика сначала проектирования и создания API, а затем создания на их основе остальной части приложения;
* **BDT - Behavior Driven Testing:** в тестировании на основе поведения ваши тесты основаны на user stories, которые описывают некоторые конкретные ожидаемые действия приложения. Вместо проверки деталей реализации вы фактически проверяете то, что важно больше всего: правильно ли приложение выполняет user stories. Еще одним преимуществом является понятность тестов для менеджеров, аналитиков и т.п.;
* **MDT - Model Driven Testing:** Тестирование на основе моделей - это метод тестирования черного ящика, при котором поведение тестируемого программного обеспечения во время выполнения проверяется на основе прогнозов, сделанных моделями. Модель - это описание поведения системы. Поведение может быть описано в виде наглядной схемы, Data Flow, Control Flow, Dependency Graphs, Decision Tables, State transition machines или mind map. Простой аналогией модели в тестировании является электрическая схема при разработке электроприбора. Этот подход к тестированию требуется, когда высока цена ошибки в большом продукте и нужно как можно раньше попытаться ее предотвратить;
* **DDT - Data Driven Testing (table-driven testing or parameterized testing):** в тестировании на основе данных тестовые данные хранятся в виде таблицы. Оно позволяет одним скриптом выполнять тесты для всех тестовых данных из таблицы и ожидать результатов теста в той же таблице;
* **VDT - Value Driven Testing:** тестирование на основе ценности - это подход, в основе которого лежит анализ ценности и экономической целесообразности тестирования.