**МОДУЛЬ 1: «ВВЕДЕНИЕ В ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ»**

ТЕСТИРОВАНИЕ. ОСНОВНЫЕ ЦЕЛИ И ЗАДАЧИ

ВВЕДЕНИЕ

**ТЕСТИРОВАНИЕ** – процесс выполнения программы с целью обнаружения факта наличия ошибок. Это классическое определение тестирования, принадлежащее Гленфорду Майерсу [Майерс 1980].

**ТЕСТИРОВАНИЕ ПРОГРАММНОГО КОДА** – процесс выполнения программного кода, направленный на выявление существующих в нем дефектов. Под дефектом понимается участок программного кода, выполнение которого при определенных условиях приводит к неожиданному поведению системы (т.е. поведению, не соответствующему требованиям).

**ТЕСТИРОВАНИЕ** – один из наиболее устоявшихся способов обеспечения качества разработки программного обеспечения.

Тестирование отвечает на вопрос “Как это сделано?” или “Соответствует ли поведение разработанной программы требованиям?”

Процесс тестирования относится к проверке корректности программной реализации системы, соответствия реализации требованиям, т.е. тестирование – это управляемое выполнение программы с целью обнаружения несоответствий ее поведения и требований.

**ЦЕЛЬ ТЕСТИРОВАНИЯ** – минимизация количества существенных дефектов в конечном продукте. Грамотно организованное тестирование дает гарантию того, что: а) система удовлетворяет требованиям; б) система ведет себя в соответствии с требованиями во всех предусмотренных ситуациях.

**ЗАДАЧА ТЕСТИРОВАНИЯ** – определение условий, при которых проявляются дефекты системы и протоколирование этих условий.

**ОТЛАДКА** – это процесс локализации и устранения ошибок.

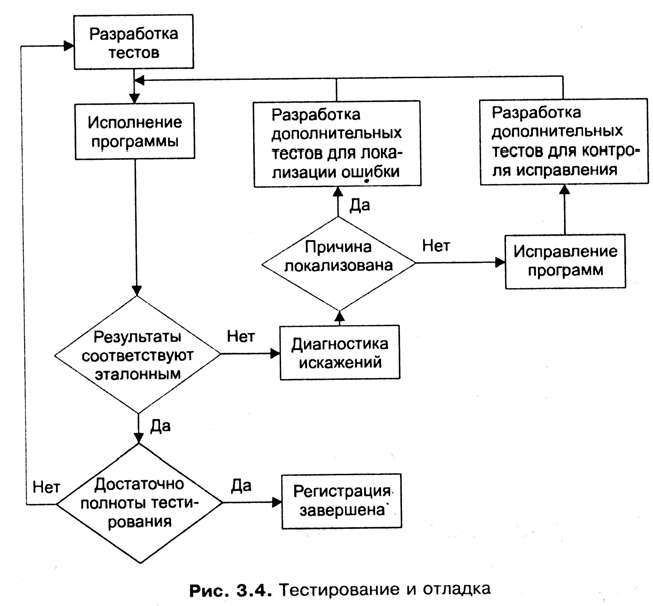
**ЗАДАЧА ОТЛАДКИ** – выявление конкретных дефектных участков программного кода и их исправление, которое выполняется по результатам тестирования.

О ПЕРВОМ ВЫЛОВЛЕННОМ БАГЕ

На компьютерном жаргоне ошибка в программе называется багом (от английского слова: bug – жучок). Происхождение этого термина связывают с мошкой, извлеченной в 1947 году из реле компьютера “Марк II”. Грейс Мюррей Хоппер приклеила ее в журнал с примечанием “первый случай выловленного бага”.

СХЕМА ПРОЦЕССА ТЕСТИРОВАНИЯ И ОТЛАДКИ

Процессы тестирования и отладки схематически могут быть представлены так, как показано на рис. 3.4 [Липаев 1986].



ТРУДОЗАТРАТЫ НА ТЕСТИРОВАНИЕ

Как одна из основных фаз процесса разработки программного продукта (Дизайн приложения ‐ Разработка кода ‐ Тестирование), тестирование характеризуется достаточно большим вкладом в суммарную трудоемкость разработки продукта. Широко известна оценка распределения трудоемкости между фазами создания программного продукта: 40%‐20%‐40%, Рис.1.1 [Котляров 2006].

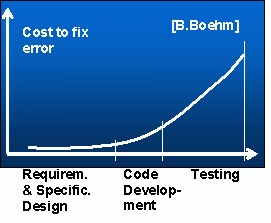


Рис. 1.1. Оценка трудоемкости обнаружения и исправления ошибок при создании программного продукта

[Boehm 1981].

Задачей ближайшего будущего является движение в сторону такого распределения трудоемкости: 60%‐ 20%‐20%, Рис.1.2 [Котляров 2006].

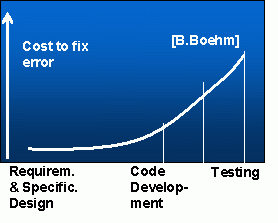


Рис. 1.2. Аналогичная оценка при автоматизации дизайна [Boehm 1981].

ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Соберем вместе основные понятия и определения, связанные с процессом тестирования [Майерс 1980].

**ТЕСТИРОВАНИЕ** – процесс выполнения программы с целью обнаружения факта наличия ошибок.

**ТЕСТОВЫЕ ДАННЫЕ** – входы, которые используются для проверки системы.

**ТЕСТОВАЯ СИТУАЦИЯ (TEST CASE)** – входы для проверки системы и предполагаемые выходы в зависимости от входов, если система работает в соответствии с ее спецификацией требований.

**ХОРОШАЯ ТЕСТОВАЯ СИТУАЦИЯ** – ситуация, которая обладает большой вероятностью обнаружения пока еще необнаруженной ошибки.

**УДАЧНЫЙ ТЕСТ** ‐ тест, который обнаруживает пока еще необнаруженную ошибку.

**ОШИБКА** ‐ действие программиста на этапе разработки, которое приводит к тому, что в программном обеспечении содержится внутренний дефект, который в процессе работы программы может привести к неправильному результату.

**ОТКАЗ** – непредсказуемое поведение системы, приводящее к неожидаемому результату, которое могло быть вызвано дефектами, содержащимся в ней.

Таким образом, в процессе тестирования программного обеспечения, как правило, проверяют следующее:

* Проверка того, что программное обеспечение соответствует требованиям на него.
* Проверка того, что в ситуациях, не отраженных в требованиях, программное обеспечение ведет себя адекватно, то есть не происходит отказ системы.
* Проверка программного обеспечения на предмет типичных ошибок, которые делают программисты.

ЖИЗНЕННЫЙ ЦИКЛ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

МОДЕЛИ ЖИЗНЕННОГО ЦИКЛА.

**ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ** – совокупность итерационных процедур, связанных с последовательным изменением состояния программного обеспечения от формирования исходных требований к нему до окончания его эксплуатации конечным пользователем.

КАСКАДНЫЙ ЖИЗНЕННЫЙ ЦИКЛ

**КАСКАДНЫЙ ЖИЗНЕННЫЙ ЦИКЛ (ВОДОПАДНЫЙ)** основан на постепенном увеличении степени детализации описания всей разрабатываемой системы. Каждое повышение степени детализации определяет переход к следующему состоянию разработки (Рис.1) [Синицын 2006].

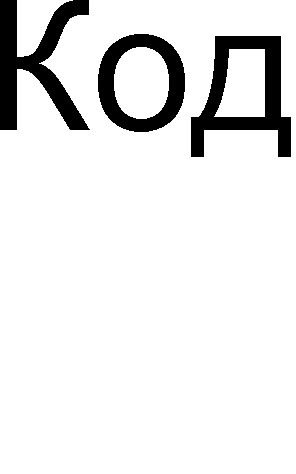
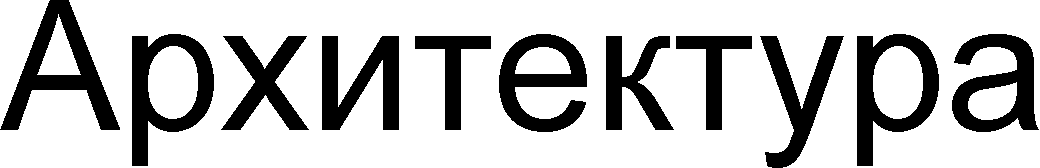
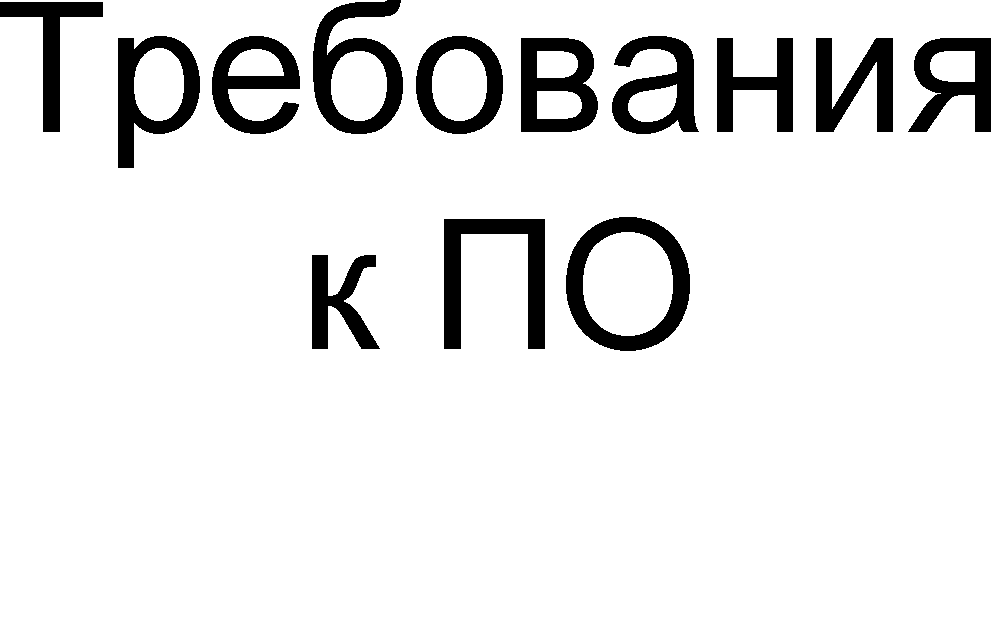
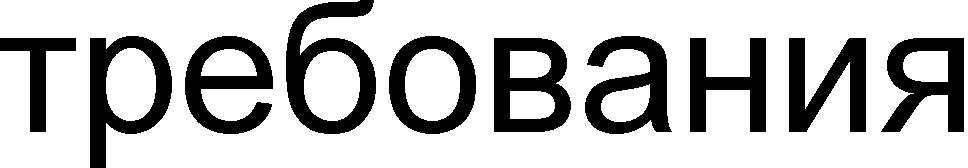
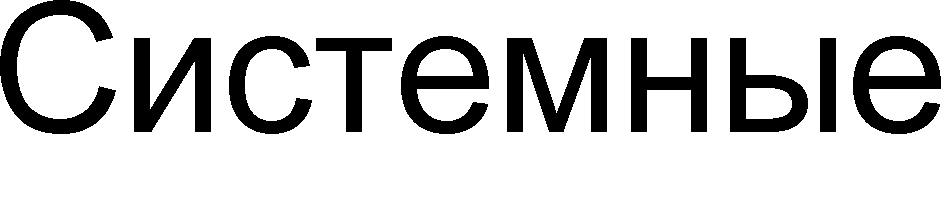


Рис.1 Каскадная модель жизненного цикла.

**ДОСТОИНСТВО КАСКАДНОГО ЖИЗНЕННОГО ЦИКЛА** – переход к следующему этапу происходит только тогда, когда **полностью** завершены все работы предыдущего этапа. То есть сначала полностью готовятся все требования к системе, затем по ним **полностью** готовятся все требования к программному обеспечению, полностью разрабатывается архитектура системы и так далее до тестирования.

**НЕДОСТАТОК КАСКАДНОГО ЖИЗНЕННОГО ЦИКЛА** – в случае больших систем работа на каждом этапе занимает значительное время, а внесение изменений в первичные документы либо невозможно, либо вызывает лавинообразное изменения на всех других этапах.

Дополнительно необходима процедура принятия решения. Если тесты обнаружили несоответствие реализации требованиям, то причина может крыться: а) в неправильном тесте; б) в ошибке кодирования; в) в неверной архитектуре системы; г) некорректности требований к программному обеспечению и т.д. Все эти случаи требуют анализа для принятия решения о том, на какой этап жизненного цикла надо возвратиться для устранения обнаруженного несоответствия.

V‐ОБРАЗНЫЙ ЖИЗНЕННЫЙ ЦИКЛ

В качестве модификации классической каскадной модели стала применяться модель жизненного цикла, содержащая процессы двух видов – основные процессы разработки, аналогичные процессам каскадной модели и процессы верификации, представляющие собой цепь обратной связи по отношению к основным процессам (Рис.2) [Синицын 2006].

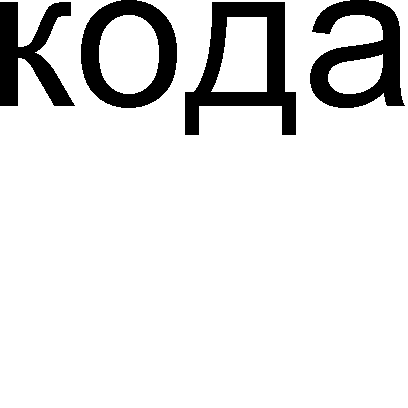
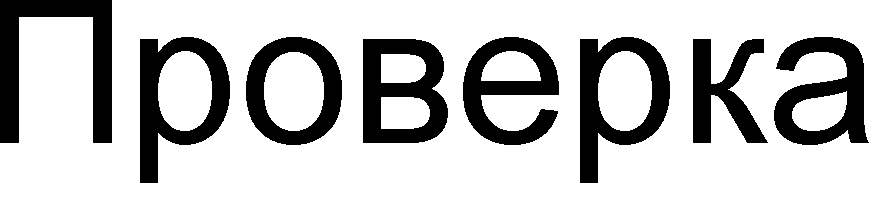
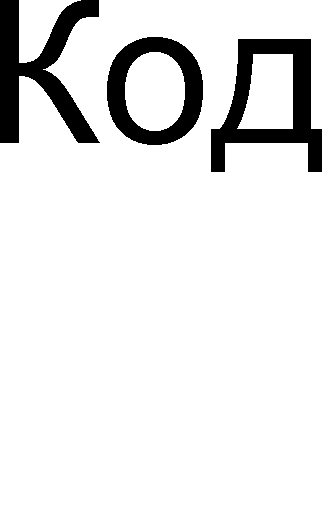
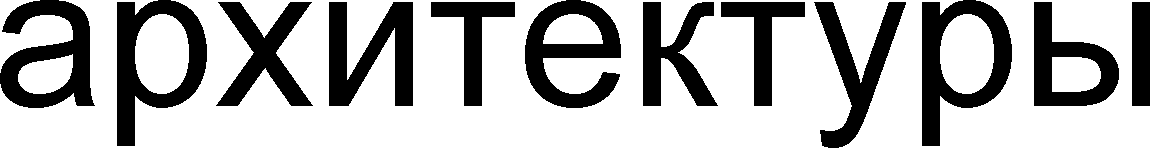
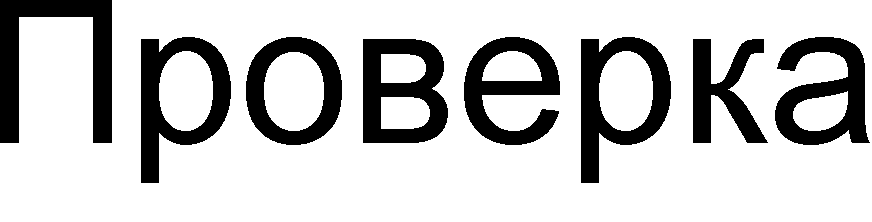
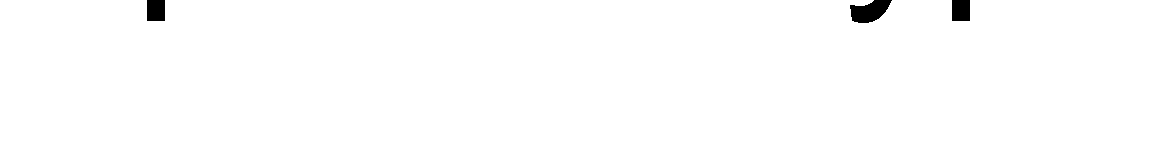
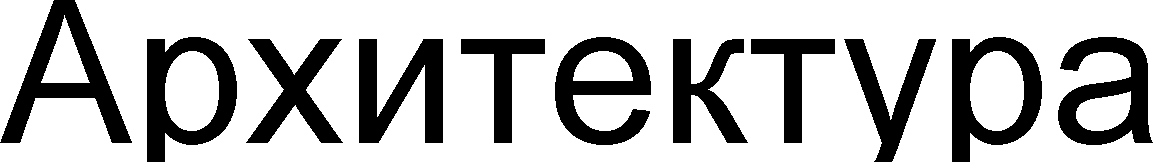
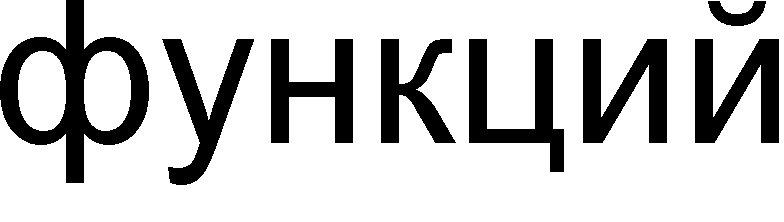
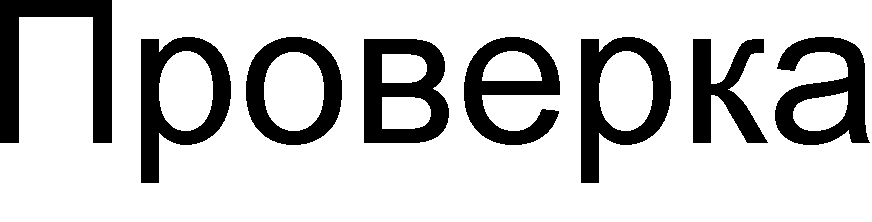
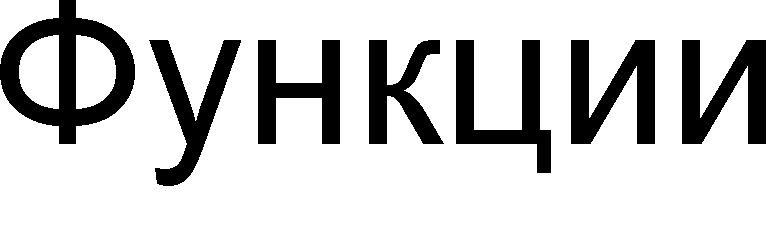
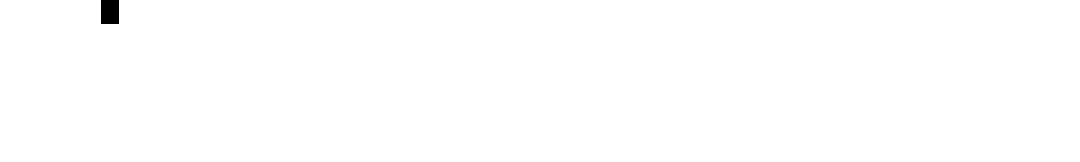
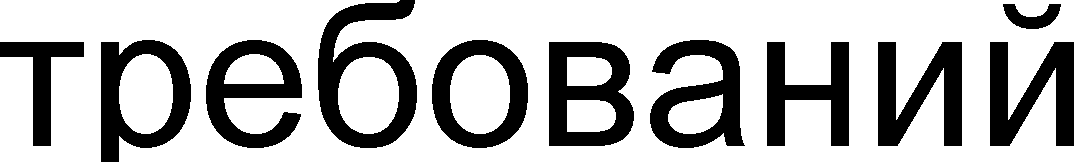
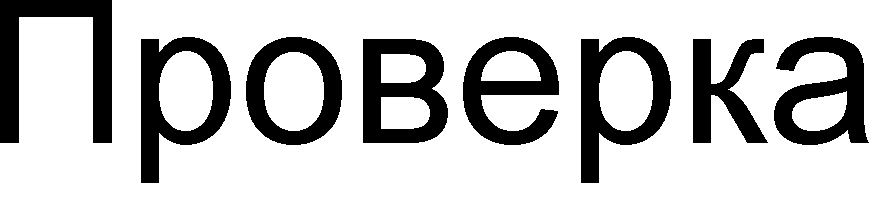
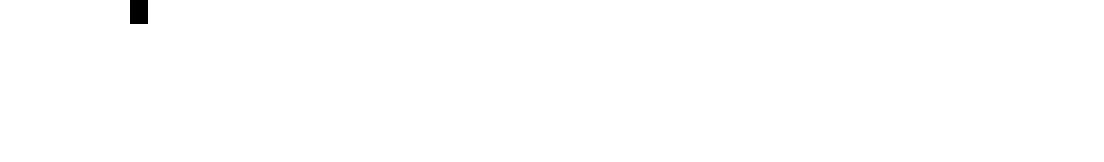
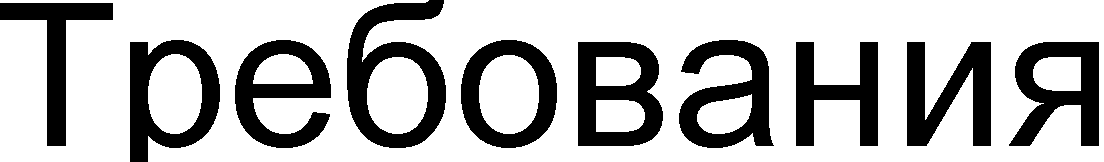


Рис.2 V‐образный жизненный цикл.

Таким образом, в конце каждого этапа жизненного цикла разработки, а зачастую и в процессе выполнения этапа, осуществляется проверка взаимной корректности требований различных уровней.

**ДОСТОИНСТВА**. Данная модель позволяет более оперативно проверять корректность разработки.

**НЕДОСТАТКИ**. Как и в каскадной модели предполагается, что на каждом этапе разрабатываются документы, описывающие поведение всей системы в целом.

СПИРАЛЬНЫЙ ЖИЗНЕННЫЙ ЦИКЛ

Оба рассмотренных выше типа жизненных циклов предполагают, что заранее известны все требования пользователей или предполагаемые пользователи системы могут высказывать свои требования к будущей системе, не видя ее перед глазами. Решение, исправляющее основной недостаток V‐образного жизненного цикла, – на каждом этапе разрабатывается как бы полное описание системы, что и реализовано в спиральной модели жизненного цикла (Рис.3) [Синицын 2006].

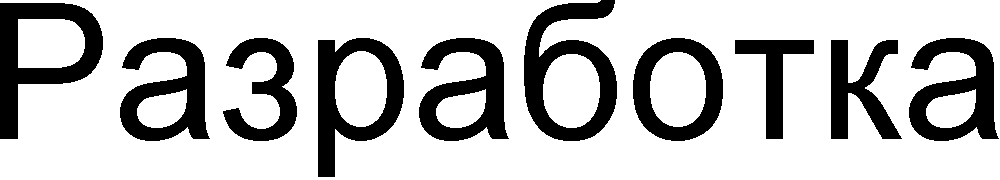
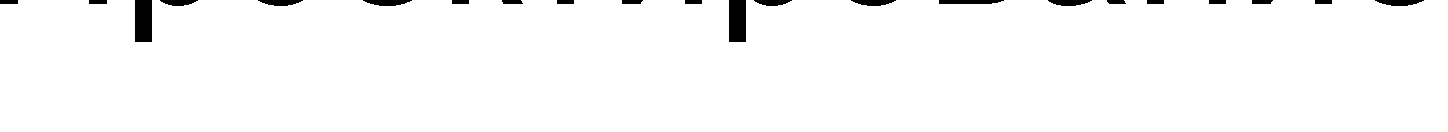
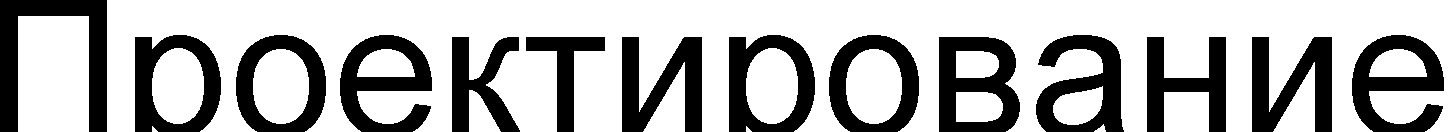
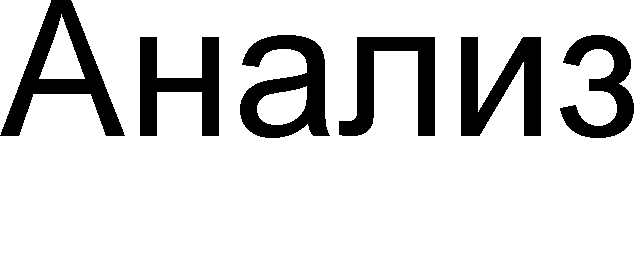
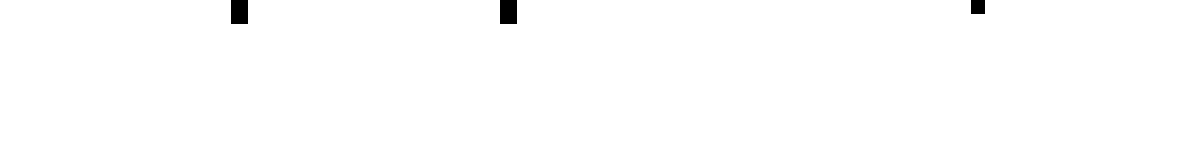
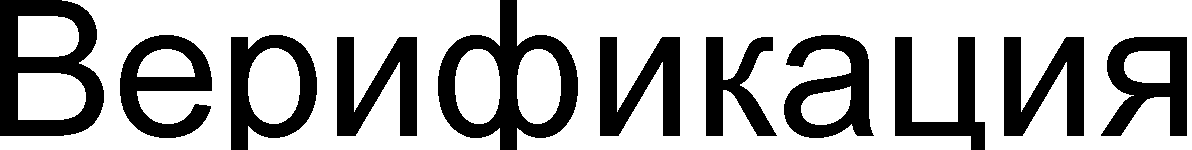


Рис.3 Спиральный жизненный цикл.

В **СПИРАЛЬНОЙ МОДЕЛИ** разработка системы происходит повторяющимися этапами – витками **СПИРАЛИ**. Каждый виток спирали – один каскадный или V‐образный жизненный цикл. В конце каждого витка получается законченная версия системы, реализующая некоторый набор функций.

Затем она предъявляется пользователю, на следующий виток переносится вся документация, разработанная на предыдущем витке, и процесс повторяется. Таким образом, система разрабатывается постепенно, проходя постоянные согласования с заказчиком. На каждом витке спирали функциональность системы расширяется постепенно, дорастая до полной функциональности.

ЭКСТРЕМАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Для систем, требования к которым изменяются часто, необходимо было еще больше уменьшить длительность витка спирального жизненного цикла. В связи с этим стали популярными быстрые жизненные циклы разработки, например жизненный цикл в методологии eXtreme Programming (XP).

**ОСНОВНАЯ ИДЕЯ ЖИЗНЕННОГО ЦИКЛА (XP)** – максимальное укорачивание длительности одного этапа жизненного цикла и тесное взаимодействие с заказчиком. На каждом этапе происходит реализация и тестирование одной функции системы, после завершения которых, система сразу передается заказчику на проверку или эксплуатацию.

**ОСНОВНАЯ ПРОБЛЕМА ДАННОГО ПОДХОДА** – интерфейсы между модулями, реализующими эту функцию. Если во всех предыдущих типах жизненного цикла интерфейсы достаточно четко определяются в самом начале разработки, поскольку заранее известны все модули, то при экстремальном подходе интерфейсы проектируются на “лету”, вместе с разрабатываемыми модулями.

СРАВНЕНИЕ РАЗЛИЧНЫХ ТИПОВ ЖИЗНЕННОГО ЦИКЛА

Особенности рассмотренных выше типов жизненного цикла сведены в таблицу 1 [Синицын 2006]. Различные типы жизненных циклов применяются в зависимости от планируемой частоты внесения изменений в систему, сроков разработки и ее сложности. Жизненные циклы с более короткими фазами больше подходят для разработки систем, требования к которым еще не устоялись и вырабатываются во взаимодействии с заказчиком системы во время ее разработки.

Таблица 1. Сравнение различных типов жизненного цикла

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип жизненного цикла** | **Длина цикла** | **Верификация и внесение изменений** | **Интеграция отдельных компонент системы** |
| Каскадный | Все этапы разработки системы.  Длинный | В конце разработки всей системы.  Редко. | Четко определенные до начала кодирования интерфейсы. |
| V‐образный | Все этапы разработки системы.  Длинный | В конце полной разработки каждого из этапов системы.  Средне. | Редко изменяемые интерфейсы. |
| Спиральный | Разработка одной версии системы.  Средний. | В конце разработки каждого из этапов версии системы.  Средне. | Периодически изменяемые интерфейсы, редко меняемые в пределах версии. |
| XP | Разработка одной истории.  Короткий. | В конце разработки каждой истории.  Очень часто | Часто изменяемые интерфейсы. |

РОЛЕВОЙ СОСТАВ ГРУППЫ ТЕСТИРОВАНИЯ

Когда проектная команда включает более двух человек, неизбежно встает вопрос о распределении ролей, прав и ответственности в команде. Практически в любом коллективе разработчиков можно выделить перечисленные ниже роли. Некоторые из них могут вовсе отсутствовать, при этом отдельные люди могут выполнять сразу несколько ролей, однако общий состав меняется мало.

**ЗАКАЗЧИК (ЗАЯВИТЕЛЬ)** – представитель организации, заказавшей разрабатываемую систему. Обычно заказчик общается только с менеджерами проекта и специалистом по сертификации или внедрению.

Заказчик имеет право изменять требования к продукту (при взаимодействии с менеджерами), читать проектную и сертификационную документацию, затрагивающую нетехнические особенности разрабатываемой системы.

**МЕНЕДЖЕР ПРОЕКТА** – обеспечивает связь между заказчиком и проектной группой. Менеджер проекта управляет ожиданиями заказчика, разрабатывает и поддерживает бизнес‐контекст проекта. Его задача – определить и обеспечить требования заказчика. Менеджер проекта имеет право изменять требования к продукту и финальную документацию на продукт.

**МЕНЕДЖЕР ПРОГРАММЫ** – обеспечивает связь внутри проектной группы, разрабатывает функциональные спецификации и управляет ими, ведет график проекта и отчитывается по состоянию проекта, инициирует принятие критичных для хода проекта решений. Менеджер программы имеет право изменять функциональные спецификации верхнего уровня, план‐график проекта, распределение ресурсов по задачам. Часто на практике роль менеджера проекта и менеджера программы выполняет один человек.

**РАЗРАБОТЧИК** – принимает технические решения, которые могут быть реализованы и использованы, создает продукт, удовлетворяющий спецификациям и ожиданиям заказчика. Он участвует в обзорах, реализует возможности продукта, участвует в создании функциональных спецификаций, отслеживает и исправляет ошибки. Разработчик имеет доступ ко всей проектной документации, включая документацию по тестированию, имеет право на изменение программного кода системы в рамках своих служебных обязанностей.

**СПЕЦИАЛИСТ ПО ТЕСТИРОВАНИЮ** – определяет стратегию тестирования, тестовые требования и тестовые планы для каждой из фаз проекта, выполняет тестирование системы, собирает и анализирует отчеты о прохождении тестирования. В реальности роль специалиста по тестированию часто разбивается на две – разработчика тестов и тестировщика. Тестировщик выполняет все работы по выполнению тестов и сбору информации, разработчик тестов – всю остальную часть работы.

**СПЕЦИАЛИСТ ПО КОНТРОЛЮ КАЧЕСТВА** – осуществляет взаимодействие с разработчиком, менеджером программы и специалистами по безопасности и сертификации с целью отслеживания целостной картины качества продукта, его соответствия стандартам и спецификациям, предусмотренным проектной документацией. Контроль качества подразумевает в первую очередь контроль самих процессов разработки и проверку их соответствия определенным в стандартах качества критериям.

**СПЕЦИАЛИСТ ПО СЕРТИФИКАЦИИ**. При разработке систем с повышенными требованиями надежности требуется сертификация системы со стороны уполномоченного органа (обычно государственного) по соответствию эксплуатационных характеристик системы заданным критериям. Специалист по сертификации приводит документацию на программную систему в соответствие требованиям сертифицирующего органа, либо участвует в процессе создания документации с учетом этим требованиям. Специалист по сертификации ответственен за все взаимодействие между коллективом разработчиков и сертифицирующим органом. Специалист по сертификации независим от проектной группы на всех этапах создания продукта.

Взаимодействие специалиста с членами проектной группы ограничивается менеджерами по проекту и по программе.

**СПЕЦИАЛИСТ ПО ВНЕДРЕНИЮ И СОПРОВОЖДЕНИЮ** – участвует в анализе особенностей площадки заказчика, на которой планируется проводить внедрение разрабатываемой системы, выполняет весь спектр работ по установке и настройке системы, проводит обучение пользователей.

**СПЕЦИАЛИСТ ПО БЕЗОПАСНОСТИ** – ответственен за весь спектр вопросов безопасности создаваемого продукта. Его работа начинается с участия в написании требований к продукту и заканчивается финальной стадией сертификации продукта.

**ИНСТРУКТОР** – отвечает за снижение затрат на дальнейшее сопровождение продукта, обеспечение максимальной эффективности работы пользователя. Инструктор собирает статистику по производительности пользователей и создает решения для повышения производительности. Инструктор принимает участие во всех обсуждениях пользовательского интерфейса и архитектуры продукта.

**ТЕХНИЧЕСКИЙ ПИСАТЕЛЬ** – подготавливает документацию к разработанному продукту, финальное описание функциональных возможностей системы. Он участвует в написании сопроводительных документов (системы помощи, руководство пользователя).

ОСОБЕННОСТИ ИНДУСТРИАЛЬНОГО ТЕСТИРОВАНИЯ

КАЧЕСТВО ПРОГРАММНОГО ПРОДУКТА И ТЕСТИРОВАНИЕ

Качество программного продукта можно оценить некоторым набором характеристик, определяющих, насколько продукт “хорошо” с точки зрения всех потенциально заинтересованных в нем сторон. Такими сторонами являются:

* Заказчик продукта.
* Спонсор.
* Конечный пользователь.
* Разработчики продукта.
* Тестировщики продукта.
* Инженеры поддержки.
* Отдел обучения.
* Продаж и т.п.

**ТЕСТИРОВАНИЕ КАК СПОСОБ ОБЕСПЕЧЕНИЯ КАЧЕСТВА***.* Тестирование, с технической точки зрения, есть процесс выполнения приложения на некоторых входных данных и проверка получаемых результатов с целью подтвердить их корректность по отношению к ожидаемому результату.

Рассмотрим пример. В качестве приложения возьмем программу для работы с сетью (browser), критерии качества которой приведены в Табл.9.1 [Котляров 2006].

Таблица 9.1. Критерии качества программы browser

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Пользователь** | **Заказчик** | **Инженер поддержки** |
| Функциональная полнота | + | ‐ | ‐ |
| Цена разработки | ‐ | + | ‐ |
| Отсутствие дефектов | + | Косвенно | + |
| Удобство использования | + | ‐ | ‐ |

|  |  |  |  |
| --- | --- | --- | --- |
| Возможность внесения изменений в будущем | ‐ | Косвенно | + |
| Легкость исправления дефектов | ‐ | ‐ | + |
| Документация на реализацию, в том числе комментарии | ‐ | ‐ | + |
| Своевременность исполнения проекта | ‐ | + | ‐ |

Матрица критериев качества заинтересованных в них участников для рассматриваемого проекта приведена в Табл.9.2 [Котляров 2006].

Таблица 9.2. Матрица критериев качества и элементов системы обеспечения качества

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Тестирование** | **Анализ рынка и**  **специальные лаборатории** | **Обзор ы кода** | **Анализ дизайна** | **Аудиты**  **процесса разработки** |
| **Полнота функциональности** | +, не всегда эффективно | + | ‐ | ‐ | ‐ |
| **Стоимость разработки** | ‐ | ‐ | ‐ | ‐ | + |
| **Отсутствие дефектов** | + | ‐ | + | ‐ | ‐ |
| **Удобство использования** | +, не всегда эффективно | + | ‐ | ‐ | ‐ |
| **Возможность внесения изменений в будущем** | ‐ | ‐ | +‐ | + | ‐ |
| **Легкость исправления дефектов** | ‐ | ‐ | + | + | ‐ |
| **Документация на реализацию, в том числе комментарии** | ‐ | ‐ | + | ‐ | + |
| **Своевременность исполнения проекта** | ‐ | ‐ | ‐ | ‐ | + |

Данные (Табл.9.1 и Табл.9.2) показывают, что из восьми элементов общего качества продукта тестирование способно оценить и контролировать только три (1, 3, 4), причем наиболее эффективно тестирование контролирует отсутствие дефектов (3).

УРОВНИ ТЕСТИРОВАНИЯ

В тестировании выделяются три основных уровня.

1. Модульное тестирование.
2. Интеграционное тестирование.
3. Системное тестирование.

ФАЗЫ ПРОЦЕССА ТЕСТИРОВАНИЯ

В процессе тестирования выделяют следующие фазы:

1. **ОПРЕДЕЛЕНИЕ ЦЕЛЕЙ**, включающее следующую конкретизацию: какие части системы будут тестироваться, какие аспекты их работы будут выбраны для проверки, каково желаемое качество и т.п.
2. **ПЛАНИРОВАНИЕ** – создание графика разработки тестов для каждой тестируемой подсистемы; оценка необходимых человеческих, программных и аппаратных ресурсов; разработка расписания тестовых циклов.
3. **РАЗРАБОТКА ТЕСТОВ**, то есть тестового кода для тестируемой системы, если необходимо – кода системы автоматизации тестирования и тестовых процедур (выполняемых вручную).
4. **ВЫПОЛНЕНИЕ ТЕСТОВ**: реализация тестовых циклов.
5. **АНАЛИЗ РЕЗУЛЬТАТОВ**.

После анализа результатов возможно повторение процесса тестирования, начиная с пунктов 3, 2 или даже

1.

ТЕСТОВЫЙ ЦИКЛ

**ТЕСТОВЫЙ ЦИКЛ** – это цикл исполнения тестов, включающий фазы 4 и 5 тестового процесса. Тестовый цикл заключается в прогоне разработанных тестов на некотором состоянии программного кода.

Тестовый цикл включает следующую последовательность действий.

1. Проверка готовности системы и тестов к проведению тестового цикла.
2. Подготовка тестовой машины в соответствии с требованиями, определенными на этапе планирования.
3. Воспроизведение среза системы.
4. Прогон тестов в соответствии с документированными процедурами.
5. Сохранение тестовых протоколов (test log).
6. Анализ протоколов тестирования и принятие решения о том прошел или не прошел каждый из тестов (Pass/Fail).
7. Анализ и документирование результатов цикла.

ТЕСТОВЫЙ ПЛАН

Тестовый план ‐ это документ, или набор документов, содержащий следующую информацию:

1. Тестовые ресурсы.
2. Перечень функций и подсистем, подлежащих тестированию.
3. Тестовую стратегию, включающую:
   * Анализ функций и подсистем с целью определения наиболее слабых мест, где появление дефектов наиболее вероятно.
   * Определение стратегии выбора входных данных для тестирования.
   * Определение потребности в автоматизированной системе тестирования и дизайн такой системы.
4. Расписание тестовых циклов.
5. Фиксацию тестовой конфигурации: состава и конкретных параметров аппаратуры и программного окружения.
6. Определение списка тестовых метрик, которые на тестовом цикле необходимо собрать и проанализировать.

ТИПЫ ТЕСТИРОВАНИЯ

Типы тестирования по виду подсистемы или продукта:

1. **ТЕСТИРОВАНИЕ ОСНОВНОЙ ФУНКЦИОНАЛЬНОСТИ**, когда тестированию подвергается система, являющаяся основным выпускаемым продуктом.
2. **ТЕСТИРОВАНИЕ ИНСТАЛЛЯЦИИ** включает тестирование сценариев первичной инсталляции системы, сценариев повторной инсталляции, тестирование деинсталляции, тестирование инсталляции в условиях наличия ошибок в инсталлируемом пакете, в окружении или в сценарии и т.п.
3. **ТЕСТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОЙ ДОКУМЕНТАЦИИ** включает проверку полноты и понятности описания правил и особенностей использования продукта, наличие описания всех сценариев и функциональности, синтаксис и грамматику языка, работоспособность примеров и т.п.

Типы тестирования по способу выбора входных значений:

1. Функциональное тестирование, при котором проверяется:
   * Покрытие функциональных требований.
   * Покрытие сценариев использования.
2. Стрессовое тестирование, при котором проверяются экстремальные режимы использования продукта.
3. Тестирование граничных значений.
4. Тестирование производительности.
5. Тестирование на соответствие стандартам.
6. Тестирование совместимости с другими программно‐аппаратными комплексами.
7. Тестирование работы с окружением.
8. Тестирование работы на конкретной платформе.

ТЕСТИРОВАНИЕ СЦЕНАРИЕВ

Разработка тестов, основанных на использовании сценариев, осуществляется по следующей методике:

1. Определяется модель использования, включающая операционное окружение продукта и "актеров".
2. Разрабатываются сценарии использования продукта.
3. Разрабатывается набор тестов, покрывающих заданные сценарии.

Использование сценариев не требует наличия полной формальной спецификации требований, но зато может потребовать больше времени на разработку и анализ. Еще одна особенность тестирования сценариев заключается в том, что этот метод направляет тестирование на проверку конкретных режимов использования продукта, что позволяет находить дефекты, которые метод тестирования по функциональным требованиям может пропустить.

СТАНДАРТЫ КАЧЕСТВА

**ПРОЦЕСС ГАРАНТИИ КАЧЕСТВА** обеспечивает проведение проверок, гарантирующих, что процесс разработки удовлетворяет набору определенных требований (стандартов), необходимых для выпуска качественной продукции. Он проверяет, что все предусмотренные стандартами разработки процедуры выполняются и при выполнении соблюдаются декларированные для них правила.

Требования, предъявляемые к организации работы, необходимой для выпуска качественной продукции, оформлены в виде стандартов качества.

Наиболее часто цитируемая и известная группа стандартов качества – серия стандартов ISO 9000.

В дополнение к ним существует стандарт, содержащий требования к жизненному циклу разработки программного обеспечения – ISO 12207.

В реальной практике сейчас наиболее широко применяется стандарт ISO 12207, а в отечественных госструктурах используются стандарты серии ГОСТ 34.

Стандарты комплекса ГОСТ 34 на создание и развитие аппаратных средств (АС) ‐ обобщенные, но воспринимаемые как весьма жесткие по структуре жизненного цикла (ЖЦ) и проектной документации.

Международный стандарт ISO/IEC 12207 на организацию жизненного цикла продуктов программного обеспечения (ПО) – содержит общие рекомендации по организации жизненного цикла, не постулируя при этом его жесткой структуры.

ЛИТЕРАТУРА

[Майерс 1980] – Г. Майерс. Надежность программного обеспечения. М.: Мир, 1980. 360 с. [Липаев 1986] – В.В. Липаев. Тестирование программ. М.: Радио и связь, 1986.

[Котляров 2006] – В.П. Котляров, Т.В. Коликова. Основы тестирования программного обеспечения. Учебное пособие. М.: Интернет‐Университет Информационных технологий. БИНОМ. Лаборатория Знаний, 2006. –285 с.

[Boehm 1981] – Software Engineering Economic. – Prentice‐Hall, Inc, N.J. 1981.

[Синицын 2006] – Синицын С.В., Налютин Н.Ю. Верификация программного обеспечения. Курс лекций. Московский инженерно‐физический институт. М. 2006.