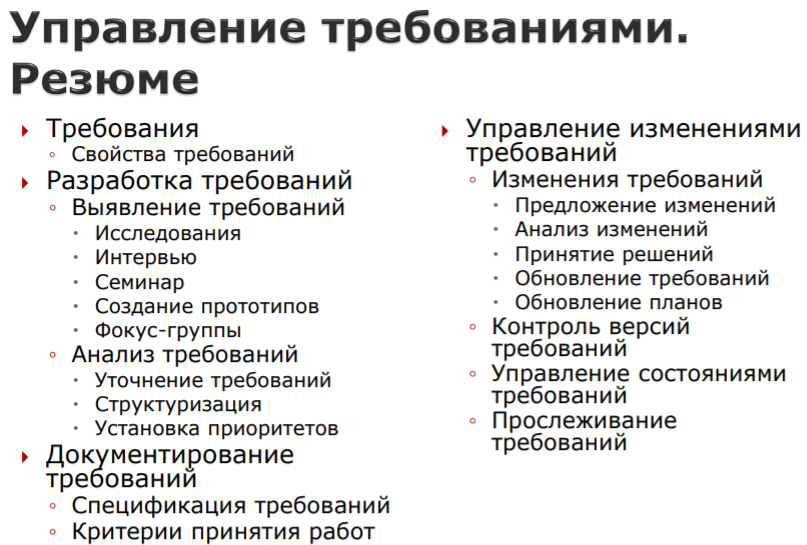
**Часто используемые картинки**

****



****

****

1. Жизненный цикл ПО. Классификация стратегий конструирования ПО.

**Жизненный цикл программного обеспечения** — это период времени, который начинается с момента принятия решения о создании программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

Основные **фазы** жизненного цикла ПО:

* Анализ и планирование
* Проектирование
* Разработка
* Тестирование
* Документирование
* Эксплуатация/Сопровождение

**Критерии успешности проекта:**

* Качество
* Реализованы все возможности
* Эти возможности реализованы с надлежащим качеством
* Время
* Проект реализован вовремя
* Этапы проекта реализованы вовремя
* Бюджет
* Проект уложился в планируемый бюджет

**Методологи проектирования ПО определяются:**

* составом и последовательностью работ
* ролью участников проекта
* составом и шаблонами документов
* организацией и управлением требованиями
* порядком контроля и проверки качества
* способами взаимодействия участников

**Характеристики методологий проектирования:**

* **стратегия конструирования**
* **адаптивость процесса**
* **этапы и связи между ними**
* **формулировка требований**

**Стратегии конструирования ПО**

◦ Однократные (водопадные) стратегии: классическая каскадная модель

Линейная последовательность этапов конструирования. Определены все требования, один цикл конструирования, промежуточных версий нет

◦ Инкрементные стратегии: инкрементная модель; RAD

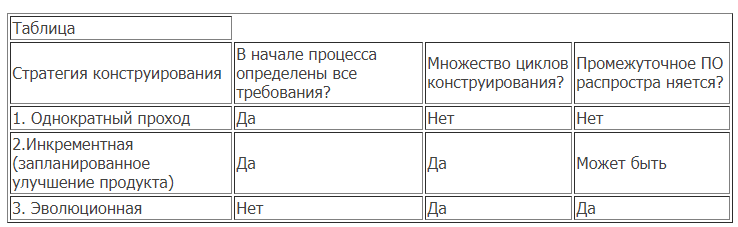
В начале процесса определяются все пользовательские и системные требования, оставшаяся часть конструирования выполняется в виде последовательности версий. Первая версия реализует часть запланированных возможностей, следующая версия реализует дополнительные возможности и т. д., пока не будет получена полная система; (множество циклов конструирования)

◦ Эволюционные стратегии: Прототипирование; Спиральная модель; Экстремальное программирование; Модель SCRUM; Lean Software Development

Система также строится в виде последовательности версий, но в начале процесса определяются не все требования. Требования уточняются в результате разработки версий. (множество циклов конструирования)

◦ Смешанные подходы: Rational Unified Process (RUP)

**Характеристики стратегий конструирования ПО в соответствии с требованиями стандарта IEEE/EIA 12207**



Рассматриваемые методологии

* Классическая (водопадная) модель – линейная, итерационная, каскадная, строгая каскадная.
* Прототипирование
* Инкрементная модель
* Быстрая разработка приложений (RAD)
* Спиральная модель
* Rational Unified Process (RUP)
* Экстремальное программирование (ХР)
* Scrum
* Lean Software Development

**Адаптивность процесса к окружению:**

1. **Прогнозирующие (тяжеловесные)**

**Фиксированные требования, большая команда, разная квалификация разработчиков**

1. **Адаптивные (облегченные)**

**Постоянно меняющиеся требования, маленькая команда, высококвалифицированные разработчики**



Выбор методологии зависит от:

* Решаемых задач
* Сроков реализации
* Команды разработчиков:
* Размер команды разработчиков
* Опыт команды разработчиков
* Сработанность команды разработчиков
* Местонахождение разработчиков
* Механизмы взаимодействия в команде разработчиков
* Заказчика:
* Требований заказчика
* Механизмов взаимодействия с заказчиком
* Квалификации заказчика
* и т.д.

2. Стратегии конструирования ПО. Классическая модель проектирования ПО

**Каскадная / Классическая / Водопадная модель** – модель процесса разработки программного обеспечения, в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки.

} Предложена в 1960-х годах, впервые описана 1970 г., У. Ройсом

} Водопадный (однократный) подход

} Относится к прогнозирующим методологиям

} Предполагает полное наличие всех требований на момент старта проекта

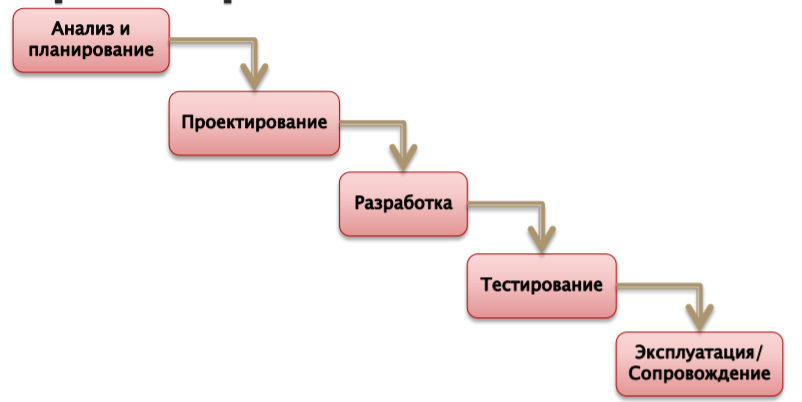
} Требования не могут меняться в процессе проектирования

} Программный продукт появляется по окончании проектирования

} Промежуточные версии не предусмотрены

Этапы:

1. Анализ и планирование – сбор и анализ требований, планирование проекта
2. Проектирование – разработка архитектуры, моделей данных, алгоритмов
3. Реализация – кодирование, отладка
4. Тестирование/верификация – отвечает ли текущее состояние разработки, достигнутое на данном этапе, требованиям этого этапа
5. Сопровождение – внедрение, эксплуатация, внесение изменений

****

В исходном виде мало подходит к современным проектам, есть модификации:

Линейная модель, итерационная (обратная связь после каждого этапа), каскадная (завершение каждого этапа проверкой, можно вернуться к любому из пройденных этапов), строгая каскадная (минимизация возвратов к пройденным этапам, можно вернуться лишь к предыдущему)

**Достоинства**

Имеется план и график по всем этапам конструирования, он упорядочен, есть богатый опыт использования. Также, данная модель понятна \*большим\* заказчикам (гос, воен, фин орг)

**Недостатки**

Не всегда соответствует реальным проектам (отсутствует гибкость), часто всех требований на начальном этапе нет (но выше написано что есть), результат доступен только в конце.

3. Стратегии конструирования ПО. Прототипирование

Применяется, когда имеются не все требования

Позволяет быстро увидеть некоторые свойства продукта – удобство, внешний вид, применимость.

Часто применяется в проектировании информационных систем, программных продуктов

Используются средства быстрой разработки приложений



**Достоинства**

Обеспечивает определение полных требований к ПО, наглядно для заказчика, позволяет заказчику рано увидеть основные параметры проекта.

**Недостатки**

По сути не является полным жизненным циклом, заказчик может принять макет за продукт, разработчик может принять макет за продукт.

4. Стратегии конструирования ПО. Инкрементная модель

**Инкрементная модель** — это метод, в котором проект проектируется, реализуется и тестируется инкрементно (то есть каждый раз с небольшими добавлениями) до самого окончания разработки.

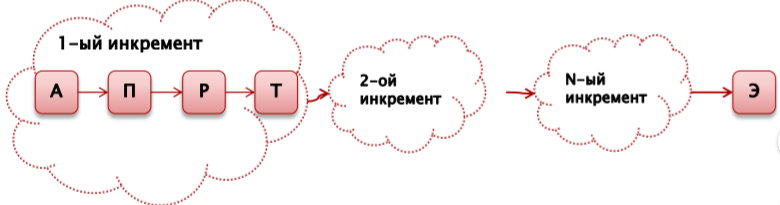
Это включает в себя как разработку, так и дальнейшую поддержку продукта. Он считается законченным в то время, когда удовлетворяет всем требованиям. Модель объединяет элементы каскадной модели с прототипированием.

} Объединяет классический подход и макетирование

} Весь проект делится на инкременты – версии продукта с определенной функциональностью

} Для каждого инкремента выполняется: ◦ Анализ ◦ Проектирование ◦ Разработка ◦ Тестирование

} Результат каждого инкремента – работающий продукт



**Достоинства**

Имеется план и график по всем этапам конструирования, промежуточные версии доступны заказчику

* Рабочее приложение выходит на ранней стадии жизненного цикла продукта
* Небольшие итерации упрощают тестирование и внесение правок
* Проще идентифицировать риски, справиться с ними
* Каждая итерация — простая в управлении контрольная точка проекта

**Недостатки**

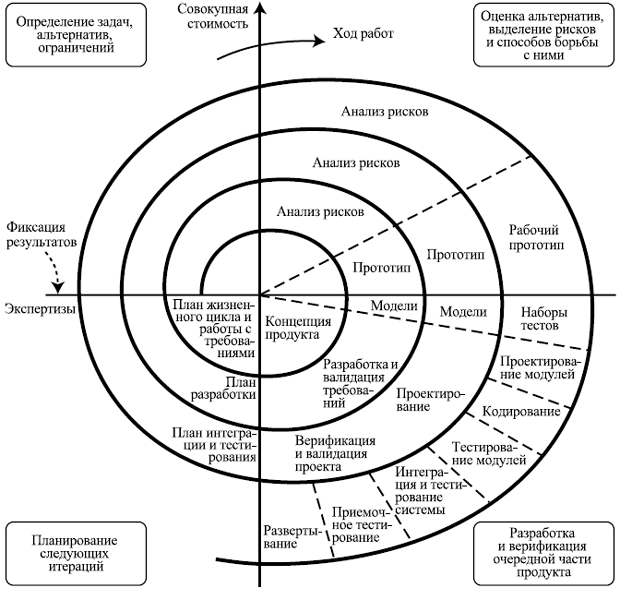
Часто всех требований на начальном этапе нет, не всегда можно заранее спланировать содержание версий, отсутствует гибкость.

**Когда использовать инкрементную модель**

* Инкрементная модель используется вкупе с четкими и понятными требованиями, которые внедряются пофазово
* Востребована при разработке веб-приложений и продуктов компаний-брендов.

5. Стратегии конструирования ПО. Спиральная модель

Спиральная модель представляет собой процесс разработки программного обеспечения, сочетающий в себе как проектирование, так и постадийное прототипирование с целью сочетания преимуществ восходящей и нисходящей концепции.



} Предложена Б. Боемом, 1988г

} Объединяет классический подход и макетирование

} Дополнена ◦ анализом рисков ◦ моделированием

Эта модель не подойдет для малых проектов, она резонна для сложных и дорогих, например, таких, как разработка системы документооборота для банка, когда каждый следующий шаг требует большего анализа для оценки последствий, чем программирование.

При итерационном подходе полезно следовать принципу «лучшее — враг хорошего». Поэтому завершение итерации долж­но производиться строго в соответствии с планом, даже если не вся запланированная работа закончена.

Планиро­вание работ обычно проводится на основе статистических дан­ных, полученных в предыдущих проектах, и личного опыта раз­работчиков. В основе спиральной модели жизненного цикла ле­жит применение прототипной — Rapid Application Development, RAD-технологии.

Эта технология обеспечивает создание на ранней стадии реализации действующей интерактивной моде­ли системы так называемой системы-прототипа, позволяющей наглядно продемонстрировать пользователю будущую систему, уточнить его требования, оперативно модифицировать интер­фейсные элементы — формы ввода сообщений, меню, выходные документы, структуру диалога, состав реализуемых функций.

} Основные компоненты ◦ Планирование ◦ Анализ ◦ Проектирование ◦ Разработка ◦ Тестирование



**Основные риски**

} Нехватка квалифицированных специалистов

} Разрыв между квалификацией специалистов и требованиями проекта

} Невыполнимые сроки

} Нереалистичный бюджет

} Непрекращающийся поток изменений

} Реализация несоответствующей функциональности

} Разработка неправильного пользовательского интерфейса

} Излишний перфекционизм в реализации деталей

} Недостаточная производительность получаемой системы

} Недостаточность информации о внешних компонентах и окружении системы

} Проблемы в работах, выполняемых внешними контрагентами (одна из сторон договора)

**Достоинства**

Адекватно отражает эволюционный характер проектирования, позволяет явно учитывать риски на каждом витке эволюции, использует моделирование и системный подход.

**Недостатки**

Высокие требования к заказчику, трудность контроля времени разработки и управления им.

6. Стратегии конструирования ПО. Быстрая разработка приложений (RAD)

**Rapid Application Development** (RAD) – это жизненный цикл процесса проектирования, созданный для достижения более высоких скорости разработки и качества ПО, чем это возможно при традиционном подходе к проектированию.

RAD предполагает, что разработка ПО осуществляется небольшой командой разработчиков за срок порядка трех-четырех месяцев путем использования инкрементного прототипирования с применением инструментальных средств визуального моделирования и разработки. Технология RAD предусматривает активное привлечение заказчика уже на ранних стадиях – обследование организации, выработка требований к системе. Причины популярности RAD вытекают из тех преимуществ, которые обеспечивает эта технология.

Наиболее существенными из них являются:

* Высокая скорость разработки;
* Низкая стоимость;
* Высокое качество.

} Инкрементная стратегия конструирования

} Использование компонентно-ориентированного конструирования

} Обеспечение очень короткого цикла разработки (60-90 дней)

} Ориентирована в основном на разработку информационных систем

**Основные этапы**

1. Бизнес-моделирование - определение списка информационных потоков между различными подразделениями.

Определяется: ◦ Какая информация создается ◦ Кто ее создает ◦ Кто ее обрабатывает ◦ Где информация применяется

1. Моделирование данных - информация, собранная на предыдущем этапе, используется для определения объектов и иных сущностей, необходимых для циркуляции информации.

По информационному потоку формируется набор объектов данных.

Определяются свойства объектов, специфицируются отношения между объектами

1. Моделирование обработки - информационные потоки связывают объекты для достижения целей разработки.

Определение преобразований объектов данных.

Создаются описания для ◦ добавления объектов данных ◦ модификации объектов данных ◦ удаления объектов данных ◦ поиска объектов данных

1. Генерация приложения - используются средства автоматической сборки для преобразования моделей системы автоматического проектирования в код.

Использование ЯП 4-го поколения. Использование готовых компонентов. Создание повторно используемых компонентов. Использования средств автоматизации

1. Тестирование и объединение

Тестирование упрощается из-за повторного использования компонентов ◦ Они не требуют автономного тестирования. Используется интеграционное тестирование

**RAD Свойства**

Область применения – проектирование информационных систем. Производительность не является критичной ◦ Неприменимо для задач реального времени. Можно привлечь достаточно разработчиков. Отсутствуют технические риски.

7. Стратегии конструирования ПО. Rational Unified Process

**Унифицированный процесс Rational** — это универсальная методология распределения задач и сфер ответственности при разработке программного обеспечения. Её цель – создание высококачественного программного обеспечения, отвечающего потребностям и запросам пользователей.

} Авторы: ◦ А. Якобсон ◦ Г. Буч ◦ Дж. Рембо } Продвигается IBM Rational } Начало разработки - 1995 г. } Первая версия RUP - 1998 г. } Наиболее глубоко проработанная методология

} Инкрементная и эволюционная итеративная методология

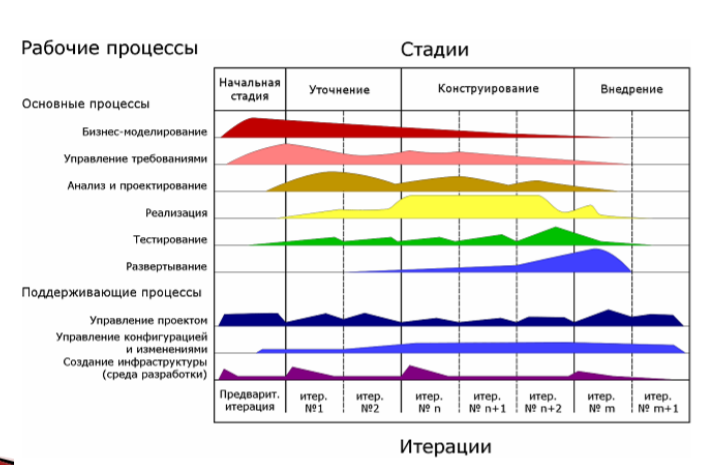
} Базируется на широком использовании UML

} На всех стадиях используются программные метрики

} Процесс делится на этапы (стадии)

} Каждый этап состоит из итераций

} Итерация – законченный цикл разработки, вырабатывающий промежуточный продукт



**Рабочие потоки процесса**

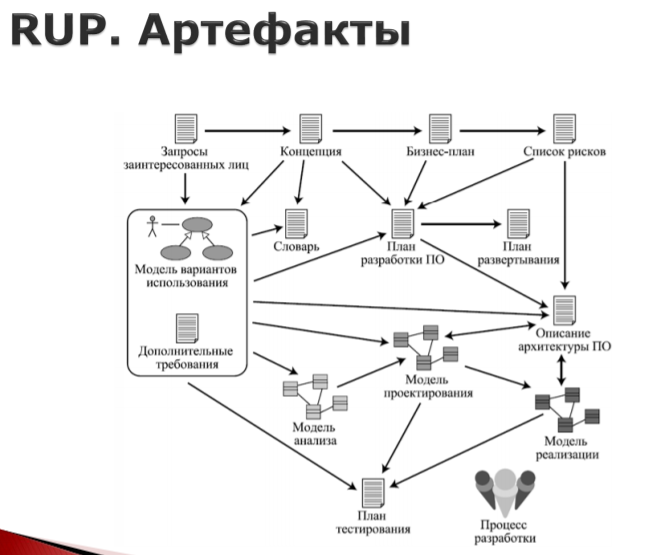
} Бизнес-моделирование

} Управление требованиями

} Анализ и проектирование: ◦ Создание статического и динамического представления системы

} Реализация ◦ Создание программного кода

} Тестирование ◦ Проверка системы в целом



1. **Начальная стадия (Inception)**

} Назначение ◦ Запуск проекта

} Цели ◦ Определение области применения ◦ Определение элементов Use Case, критических для системы ◦ Определение общих черт архитектуры ◦ Определение общей стоимости и плана проекта ◦ Идентификация основных элементов риска

**Действия**

} Формулировка области применения проекта ◦ Выявление требований и ограничений

} Планирование ◦ Подготовка основного плана развития и альтернатив развития для управления риском ◦ Определение персонала ◦ Определение проектного плана ◦ Определение зависимостей между стоимостью, планированием и полезностью

} Синтез предварительной архитектуры ◦ Развитие решений проектирования ◦ Определения используемых компонентов (разработка, покупка, повторное использование)

**Артефакты**

} Спецификация основных проектных требований

} Начальная модель Use Case (20%)

} Начальный словарь проекта

} Начальный план развития

} Начальная оценка риска

} Проектный план с этапами и итерациями

1. **Уточнение (Elaboration)**

} Назначение ◦ Создать архитектурный базис

} Цели ◦ Определение оставшихся требований (Функциональные требования выражаются с помощью Use Case) ◦ Определение архитектурной платформы системы ◦ Отслеживание рисков, устранение наибольших рисков ◦ Разработка плана итераций этапа «Конструирование»

**Действия**

} Развитие спецификации

} Формирование критических элементов Use Case, задающих дальнейшие решения

} Развитие архитектуры, выделение ее компонентов

**Артефакты**

} Модель Use Case (80%)

} Дополнительные (том числе нефункциональные) требования

} Описание программной архитектуры

} Действующий архитектурный макет

} Переработанный список элементов рисков и основной план развития

} План разработки всего проекта, включающий все итерации и критерий развития для каждой итерации

1. **Конструирование (Construction)**

} Назначение ◦ Создание программного продукта с начальной функциональностью

} Цели ◦ Минимизация стоимости разработки ◦ Быстрое получение требуемого качества ◦ Быстрое получение версий

**Действия**

} Управление ресурсами, контроль ресурсов

} Оптимизация процессов

} Полная разработка компонентов и их тестирование

} Оценивание реализаций продукта

**Артефакты**

} Программный продукт, пригодный для отчуждения от разработчиков (альфа-, бета-версия и т.п.)

} Описание текущей реализации

} Руководство пользователя

1. **Внедрение (Transition)**

} Назначение ◦ Отдать программный продукт пользователям ◦ Завершить выпуск продукта } Действия в каждой итерации ◦ Выпуск бета-версий или релизов ◦ Исправление найденных в процессе бетатестирования ошибок

} Результат ◦ Законченный продукт

**Выводы**

} Наиболее продуманная методология

} Подходит для больших и очень больших проектов (реже средних)

} Требует высокой квалификации участников

8. Стратегии конструирования ПО. Экстремальное программирование

} Экстремальное программирование (XP) } Автор – Кент Бек, 1999 г } Ориентирован на группы до 10 чел. } Группа размещается в одном помещении } Процесс: ◦ гибкий и динамичный ◦ наиболее пригоден для проектов с изменяющимися требованиями ◦ итеративный

} Основные действия: ◦ Кодирование ◦ Тестирование ◦ Выслушивание заказчика ◦ Проектирование

Основными принципами являются:

* **Итеративность.** Разработка ведется короткими итерациями при наличии активной взаимосвязи с заказчиком. Итерации как таковые предлагается делать короткими, рекомендуемая длительность – 2-3 недели и не более 1 месяца. За одну итерацию группа программистов обязана реализовать несколько свойств системы, каждое из которых описывается в пользовательской истории. Отсутствие формализации описания входных данных проекта в XP стремятся компенсировать за счет активного включения в процесс разработки заказчика как полноправного члена команды.
* **Простота решений.** Принимается первое простейшее рабочее решение. Экстремальность метода связана с высокой степенью риска решения, обусловленного поверхностностью анализа и жестким временным графиком. Реализуется минимальный набор главных функций системы на первой и каждой последующей итерации; функциональность расширяется на каждой итерации.
* **Интенсивная разработка малыми группами** (не больше 10 человек) **и парное программирование** (когда два программиста вместе создают код на одном общем рабочем месте), активное общение в группе и между группами. Все это нацелено на как можно более раннее обнаружение проблем (как ошибок, так и срыва сроков). Парное программирование направлено на решение задачи стабилизации проекта. При применении XP методологии высок риск потери кода по причине ухода программиста, не выдержавшего интенсивного графика работы. В этом случае второй программист из пары играет роль «наследника» кода. Немаловажно и то, как именно распределены группы в рабочем пространстве – в XP используется открытое рабочее пространство, которое предполагает быстрый и свободный доступ всех ко всем.
* **Обратная связь с заказчиком**, представитель которого фактически вовлечен в процесс разработки.
* **Достаточная степень смелости и желание идти на риск.**

**Методы**

1. Planning game (Игра в планирование)

} Заказчик: ◦ Объем работ ◦ Приоритет ◦ Композиция версий ◦ Сроки выпуска версий

} Разработчик: ◦ Временные оценки ◦ Последствия ◦ Процесс ◦ Подробный график работ

Вся команда разработчиков собирается вместе, принимается коллективное решение о том, какие свойства системы будут реализованы в ближайшей итерации. Трудоемкость реализации каждого свойства определяется самими программистами.

1. Small releases (Небольшие версии)

} Быстрый запуск простой системы } Версия маленькая, насколько это возможно } Версия должна быть завершенной } Обычно выпуск версии 1 раз в 2 недели

Минимальная итерация – один день, максимальная – месяц; чем чаще осуществляются релизы, тем больше недостатков системы будет выявлено. Первые релизы помогают выявить недостатки на самых ранних стадиях, далее функциональность системы расширяется на основании ПИ. Поскольку пользователь включается в процесс разработки с первого релиза, то он оценивает систему и выдает пользовательскую историю и замечания. На основании этого определяется следующая итерация, то есть, каким будет новый релиз. В XP все направлено на обеспечение непрерывной обратной связи с пользователями.

1. Metaphor (Метафора)

} Глобальное «видение» проекта, понятное всем } «Замена» большой архитектуры } В данном контексте – основная идея проекта, сведения об архитектуре

1. Simple design (Простой дизайн)

Правильный дизайн: } Выполняются все тесты } Нет дублирующей логики } Выражаются все важные идеи } Минимальное число классов и методов Добавляется в дизайн то, что нужно, только тогда, когда нужно.

Любое свойство системы должно быть реализовано как можно проще. Программисты в XP-команде работают под девизом: «Ничего лишнего!». Принимается первое простейшее работающее решение, реализуется необходимый уровень функциональности на данный момент. Тем самым экономится время программиста.

1. Testing (Тестирование)

} Для любой возможности существуют автоматические тесты } Модульные тесты – разработчики } Функциональные тесты – заказчики } Репозиторий тестов постоянно увеличивается } Код разрабатывается вместе с тестами (или после тестов)

В отличие от большинства остальных методологий тестирование в XP – одно из важнейших составляющих. Экстремальный подход предполагает, что тесты пишутся **до написания кода**. Каждый модуль обязан иметь unit test – тест данного модуля. Таким образом, в XP осуществляется регрессионное тестирование, «неухудшение качества» при добавлении функциональности. Большинство ошибок исправляются на стадии кодирования. Тесты пишут сами программисты, любой из них имеет право написать тест для любого модуля. Еще один важный принцип: тест определяет код, а не наоборот (test-driven development), то есть кусок кода кладется в хранилище тогда и только тогда, когда все тесты прошли успешно, в противном случае данное изменение кода отвергается.

1. Refactoring (Рефакторинг)

} Изменение программы для упрощения добавления новой возможности } Изменение программы после добавления новой функциональности } Программы до и после рефакторинга функционально эквивалентны

Это оптимизация существующего кода с целью его упрощения, Такая работа должна вестись постоянно. Сохраняя код прозрачным и определяя его элементы всего один раз, программисты сокращают число ошибок, которые впоследствии придется устранять. При реализации каждого нового свойства системы программист должен подумать над тем, можно ли упростить существующий код и как это поможет реализовать новое свойство. Кроме того, нельзя совмещать рефакторинг с дизайном: если создается новый код, рефакторинг следует отложить.

1. Pair programming (Парное программирование)

} Разработчики работают парами } Партнер с клавиатурой и мышью думает о реализации } Партнер без клавиатуры и мыши думает стратегически: ◦ Сработает ли данный подход? ◦ Какие еще тесты нужно разработать? } Состав пар меняется динамически } Эффективность ПП очень высокая

1. Collective ownership (Коллективное владение)

} Код – общая собственность } При необходимости код модифицируется немедленно, независимо от авторства кода

Обязательное правило: если программист внес изменения и система после этого работает некорректно, то именно этот программист должен исправить ошибки.

1. Continuous integration (Непрерывная интеграция)

} Код интегрируется раз в несколько часов. Не реже 1-го раза в день } Интеграция нового кода заканчивается после прохождения системой всех тестов } Ответственны за интеграцию пара, которая внесла изменения.

Интеграция новых частей системы должна происходить как можно чаще, как минимум раз в несколько часов. Основное правило интеграции следующее: интеграцию можно производить, если все тесты проходят успешно. Если тесты не проходят, то программист должен либо внести исправления и тогда интегрировать составные части системы, либо вообще не интегрировать их. Правило это – жесткое и однозначное. Если в созданной части системы имеется хотя бы одна ошибка, то интеграцию производить нельзя. Частая интеграция позволяет быстрее получить готовую систему, вместо того чтобы тратить на сборку неделю.

1. 40-hour week (40-часовая неделя)

} Сверхурочные – крайне нежелательны } Если постоянно требуется переработка – неправильно организован проект } Отпуск – обязателен

1. On-site customer (Локальный заказчик)

} В составе команды – представитель заказчика } Представитель – пользователь системы } Представитель отвечает на вопросы разработчиков и расставляет мелкие приоритеты

1. Coding standards (Стандарты кодирования)

} Единый стандарт кодирования } Стандарт должен способствовать коммуникациям } Стандарт должен быть добровольно воспринят командой

9. Стратегии конструирования ПО. Методология SCRUM

} Предложена в 1995, Оксфорд } Scrum – схватка } Управление хаосом } Итерационный процесс } Применима к любым этапам и особенностям разработки (в основном – разработка и сопровождение) } Хорошо стыкуется с использованием объектно-ориентированного подхода

**Артефакты**

} Backlog ◦ Список работ, которые необходимы выполнить } Backlog sprint ◦ набор требований, которые могут быть реализованы за один этап (спринт)

**Планирование**

} Спринт (Sprint) ◦ 30-тидневный (обычно) промежуток за который выполняется реализация заданной функциональности

} Планирование спринта ◦ Происходит в начале спринта

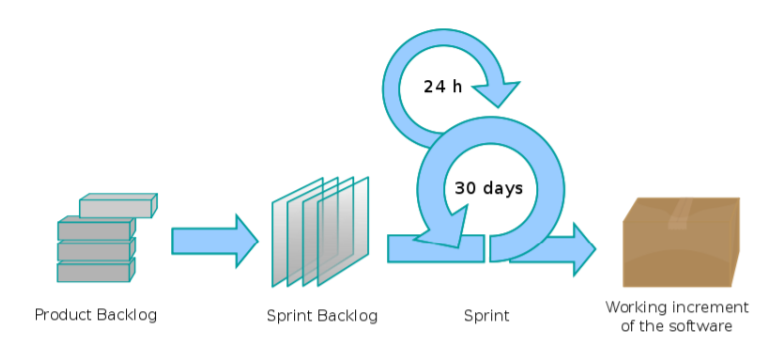
} Scrum ◦ Ежедневная встреча разработчиков

} Демонстрация ◦ Происходит в конце спринта

**Роли**

} Основные ◦ Владелец продукта (Product Owner) ◦ Руководитель (ScrumMaster) ◦ Команда (Scrum Team)

} Остальные ◦ Пользователи ◦ Представители заказчика ◦ Эксперты-консультанты



**Методология**

} Заказчик определяет и периодически меняет функциональные требования

} Владелец продукта расставляет приоритеты

} Формируются небольшие группы (1-6, реже до 9) человек для реализации небольших частей проекта

} Формируется backlog проекта

} Формируется sprint backlog для каждой группы

} Выполнение sprint происходит группой автономно. Руководство не вправе влиять на sprint

} Каждая группа ежедневно выполняет схватки (scrum) (10-30 мин): ◦ Что сделано каждым в предыдущий день? ◦ Что будет сделано каждым в следующий день? ◦ Что мешает работать или повышать производительность?

} Участвовать могут все, говорить только основные участники

} Задача руководителя группы – решать проблемы

} По окончании спринта – встреча с руководителями и заказчиками

10. Стратегии конструирования ПО. Бережливая разработка приложений

} Lean software development } Основана на Toyota Production System (TPS) } Авторы М. Поппендик и Т. Поппендик } Предложена в 2003 году

} Бережливость заключается в сокращении потерь

} Потери: ◦ Потери при разработке ПО (Лишние функциональные возможности) ◦ Избыточное проектирование (Overengineering) ◦ Незавершенные работы ◦ Поиск и исправление ошибок ◦ Ожидание ◦ Избыточные процессы ◦ Дефекты

**Принципы**

} Исключение или уменьшение потерь

} Постоянное обучение ◦ Активная обратная связь с заказчиком

} Позднее принятие решений ◦ Не прогнозы, а факты

} Быстрая доставка ◦ Короткие итерации с передачей версий

} Мотивация команды ◦ Люди – не ресурс

} Интегрирование ◦ Целостная архитектура. Понимание архитектуры заказчиком

} Целостное видение ◦ Разделение разработчиками принципов бережливости ◦ Видение проекта как единое целое

11. Управление требованиями. Требования в программных проектах. Проблемы определения требований.

Разработка требований – самая сложная часть проектирования ПО: } Требования пользователей постоянно меняются } Требования бывают неясны, двусмысленны и противоречивы, неполны } Пользователи могут быть недостаточно представительны } Получаемые спецификации не достаточно детализированы

Требование по IEEE 1990: } Условие или возможность, необходимые пользователю для решения его задач или достижения цели. } Условие или возможность, которым должна отвечать или которыми должна обладать система или ее компонента, чтобы удовлетворить контракт, стандарт, спецификацию или иной формальный документ. } Документированное представление условия или возможности, указанное в (1) или (2)

**Свойства**

1. Корректность - Требования корректны, если правильно отражают пожелания заказчика

2. Однозначность - Требования однозначны, если их формулировка подразумевает только одну трактовку

3. Полнота - Требования полны, если все пожелания заказчика нашли отражения в списке требований

4. Непротиворечивость - Требования непротиворечивы, если нет нескольких требований, которые явно или неявно противоречат друг другу

5. Приоритезация - Требования приоритезированы, если их можно разделить на несколько классов приоритета

6. Проверяемость - Требования проверяемы, если они сформулированы таким образом, что каждое можно проверить и убедиться, выполнено оно или нет

7. Модифицируемость - Требования модифицируемы, если они сформулированы таким образом, что предполагают относительно простую их модификацию

8. Отслеживаемость - Требования отслеживаемы, если они сформулированы таким образом, что по каждому требованию можно проследить: ◦ реализовано ли оно ◦ где конкретно реализовано ◦ оттестировано ли оно

**Виды требований:**

1. Функциональные требования

} Определяют набор функций, которые необходимо реализовать } Отвечают на вопрос «Что надо сделать?»

} Бизнес-требования - ◦ Формулируются заказчиками ◦ Описывают цели, которые требуется достичь с данной системой

} Пользовательские требования ◦ Какие задачи можно решить с помощью системы

} Функциональные требования ◦ Определяют требуемую функциональность

1. Нефункциональные требования

} Отвечают на вопрос «Как это должно быть сделано?»

} Характеристики качества ◦ Требования к надежности ◦ Требования к совместимости ◦ Требования к эффективности ◦ Требования к гибкости ◦ Требования к эргономике

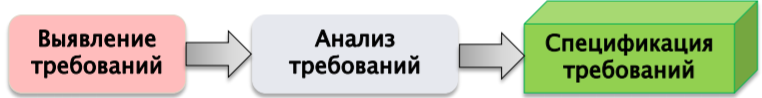
} Ограничения ◦ Соответствия стандартам и правилам ◦ Предопределенные архитектурные решения ◦ Бюджет ◦ Сроки.

} Мы сделаем проект: ◦ Быстро ◦ Качественно ◦ Недорого } Выберите 2 из 3-х

**ТРЕБОВАНИЯМИ НЕ ЯВЛЯЮТСЯ**

} Детали архитектуры } Детали реализации } Сведения о планировании } Сведения о тестировании } Проектная информация: ◦ Инфраструктура разработки ◦ Процесс разработки ◦ Команда разработки

12. Разработка требований. Выявление и анализ требований.



**Выявление требований**

} Заинтересованные лица ◦ Заказчики ◦ Менеджеры ◦ Пользователи ◦ Разработчики ◦ Служба поддержки ◦ Другие лица

} ВАЖНО: заказчик ≠пользователь

} Способы выявления требований

1. Семинары

} Открытое совместное обсуждение стоящих задач

} Заказчик ◦ излагает имеющиеся требования ◦ отвечает на вопросы исполнителя

} Исполнители ◦ слушают требования заказчика ◦ задают уточняющие вопросы ◦ документируют требования

1. Интервью

} Целенаправленный опрос-интервью

} Заказчик ◦ отвечают на вопросы из анкеты исполнителя

} Исполнители ◦ готовят специализированный вопросник (анкету) ◦ задают вопросы в соответствии с анкетой ◦ документируют ответы заказчика ◦ поясняют заказчику детали

1. Создание прототипов

} Обсуждение стоящих высокоуровневых задач

} Заказчик ◦ просмотр прототипа и внесение уточнений

} Исполнители ◦ создание прототипа ◦ демонстрация прототипа ◦ исправление прототипа ◦ документирование финального прототипа

1. Исследования

} Заказчик ◦ предоставление общего видения продукта ◦ корректировка и утверждение предложения исполнителя

} Исполнители ◦ получение от заказчика общего представления о продукте ◦ исследование рынка с целью выявления востребованных характеристик продукта ◦ анализ конкурирующих решений ◦ документирование актуальных свойств проектируемого продукта ◦ согласование результата с заказчиком

1. Фокус-группы

--------------------------------------

} Проблемы выявления требований: ◦ используемая терминология ◦ несоответствия ожиданиям пользователей ◦ неумение оценить противоречивые требования ◦ недостаточные требования ◦ неумение понять требования пользователей ◦ сложности формулирования требований ◦ использование неявных допущений ◦ использование предвзятых решений

**Анализ требований**

} Выявление требований – расходящийся процесс, цель которого собрать как можно больше данных

} Анализ требований – сходящийся процесс: ◦ уточняет данные ◦ устанавливает приоритеты ◦ структурирует информацию

} Результат анализа – спецификация требований

1. Уточнение

} Каждое требование должно быть максимально полным } Уточнение достигается путем повторных встреч с заинтересованными лицами } Не должно появляться много новых требований – иначе следует вернуться к выявлению } На этапе уточнения требования должны быть описаны количественно, а не качественно, как на этапе выявления

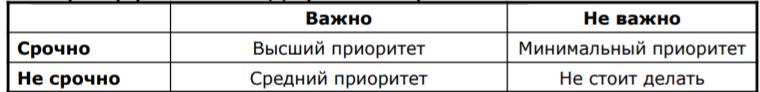
1. Приоритезация

} Необходимо отсортировать требования по важности и срочности } Должны участвовать все заинтересованные лица проекта: ◦ заказчики ◦ пользователи ◦ разработчики } Все требования не могут быть основными } Приоритеты могут изменяться по мере развития проекта

} Каждое требование относится к какой-либо качественной категории по важности: ◦ высокая, средняя, низкая ◦ обязан, должен бы, мог бы ◦ основной, полезный, желаемый

} Каждое требование относится к какой-либо качественной категории по срочности: ◦ прямо сейчас, чуть позже, когда-нибудь ◦ срочно, чуть позже, потом

} Сортируются по двум измерениям:



13. Документирование и организация требований

} Как документировать разные требования?

} Требования пользователей → пользовательские истории → варианты использования

} Бизнес-требования → документ о представлении/границах проекта

} Функциональные требования → спецификация требований к ПО

**Организация**  
} Группирование требований ◦ требования объединяются в родственные группы

} Иерархическая структуризация требований ◦ подчинение ◦ уточнение

**Способы документирования**

} Документы на естественном языке

} Графические модели ◦ диаграммы ◦ графы ◦ схемы ◦ потоки

} Формальные спецификации

14.Документирование требований. Виды документов.

1. Спецификация требований

} Фундамент всего последующего планирования, проектирования, реализации проекта } Основание для тестирования проекта } Основание для документирования проекта } Должна содержать ограничения проекта } НО: не должна содержать деталей проектирования, реализации, тестирования и управления проектом } Является исходным техническим соглашением между заказчиком и разработчиком

} Существуют различные государственные, отраслевые и корпоративные стандарты

} Наиболее распространенные в России: ◦ ГОСТ 19.201-78. ЕСПД. Техническое задание. Требования к содержанию и оформлению ◦ ГОСТ 34.602-89 «Техническое задание на создание автоматизированной системы» ◦ IEEE 830-1998 «Recommended Practice for Software Requirements Specification» (SRS)

} Шаблон не должен являться догмой (если это не требование заказчика) } Следует при необходимости модифицировать шаблон в соответствии с природой и потребностями проекта

1. Критерии принятия работ

} Должны быть приняты всеми заинтересованными лицами } Должны быть четкими и недвусмысленными } Разделы методики принятия работы должны определяться количественными параметрами, а не качественными

**Стандарты**

} ГОСТ 19.301-79 «Программа и методика испытаний. Требования к содержанию и оформлению» ◦ Программа (состав и порядок) испытаний ◦ Методика (методы) испытаний } Протокол испытаний

Тестирование требований

15.Изменения требований.

1. Причины изменения требований

} Заказчик ◦ Не понравилось после просмотра ◦ Передумал ◦ Забыл

} Рынок ◦ Такой продукт уже не продать ◦ Нужно выйти на рынок прямо сейчас, иначе этот продукт не продать

} Разработка ◦ Неточное определение границ проекта ◦ Требования плохо определены ◦ Требования не были поняты или были поняты неправильно ◦ Сработали архитектурные риски

1. Условия возможности изменений

} Водопадные стратегии – практически невозможно

} Инкрементные стратегии – возможно с некоторыми ограничениями

} Эволюционные стратегии - возможно

1. Политика управления изменениями

} Должен быть принят процесс контроля за изменениями

} Все изменения должны следовать процессу или не рассматриваться

} Для неутвержденных требований не выполняется никаких действий, кроме исследования осуществимости

} Все запросы на изменение должны быть одобрены советом по управлению изменениями } Содержание запроса на изменение должно быть доступно всем заинтересованным лицам проекта

} Начальный текст запроса должен быть неизменным

} Анализ воздействия должен проводиться для каждого изменения

} Каждое одобренное изменение (добавленное требование) должно прослеживаться до запроса на изменение

} Обоснование каждого одобрения на изменение должно быть задокументировано

1. Анализ влияния изменения

} Выявление последствий внесения изменения

} Определение всех артефактов (файлы, модели, документы, шаблоны, …), которые нуждаются в модификации, если изменение будет принято

} Определение задач, необходимых для реализации изменения } Оценка усилий для завершения этих задач

} Оценка нахождения этих задач на критическом пути проекта

} Оценка влияния на график работ

} Оценка влияния на стоимость

} Оценка приоритета изменения, учитывая ◦ Достоинства ◦ Недостатки ◦ Затраты ◦ Риски

1. Принятие/непринятие изменений

} Согласие

◦ С изменением сроков работ Формирование нового графика работ

◦ Без изменения сроков работ Откладывание низкоприоритетных требований Привлечение дополнительных сотрудников Организация краткосрочной сверхурочной работы Пожертвование качеством

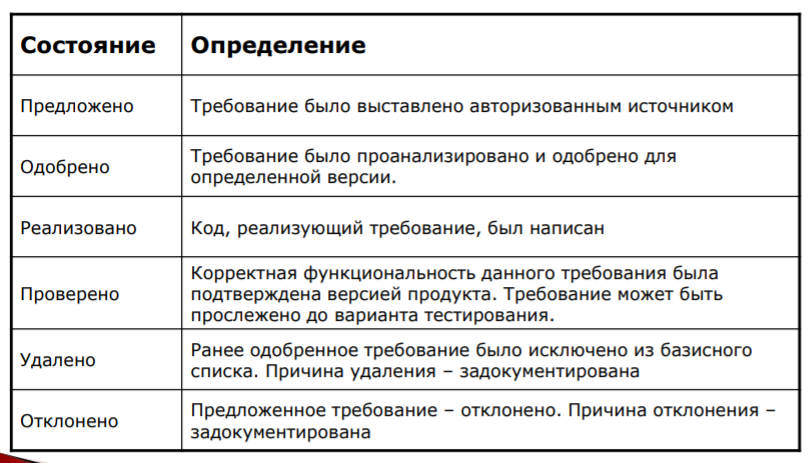
} ОТКАЗ!

16. Планирование и управление требованиями. Прослеживание требований.

} Требования могут устаревать } Требования могут быть противоречивыми

} Контроль версий документов ◦ С помощью любой системы контроля версий

} Контроль версий требований ◦ Создание начальных версий требований ◦ Ведение истории изменений ◦ Авторизованный доступ к изменениям требований



**Отслеживание состояний требований**

} Показатель прогресса проекта } Используется при анализе изменений } Обосновывает некоторые решения, принятые во время разработки } Обычно измеряется в процентах завершенности работ ◦ Часто может вводить в заблуждение

**Прослеживание требований**

} Цели: ◦ Получить подтверждение, что цели были реализованы ◦ Убедиться, что требования были оттестированы ◦ Иметь трассы всех требований от заказчика до тестовых случаев

} Необходимо прослеживать 4 типа связей:

◦ Потребности заказчика с разработанными требованиями

◦ Требования с исходными потребностями заказчиками

◦ Требования с определенными элементами продукта

◦ Определенные элементы продукта с соответствующими требованиями

****

17. Ресурсы в программных проектах. Управление ресурсами

} Ресурс – объект проекта, подлежащий управлению и планированию

} Виды ресурсов: ◦ Сотрудники ◦ Рабочее время ◦ Оборудование ◦ Машинное время ◦ Программное обеспечение

}Рабочее время

◦ Может являться атрибутом связи «сотрудник-роль»

◦ Должно учитываться при формировании команды: Нестандартное время работы Выходные Сверхурочные Отпуска

◦ В общем случае является внешним ограничением при решении задачи планирования

} Оборудование, машинное время и ПО

◦ Варианты Специализированное оборудование / ПО для разработки проекта Специализированное оборудование / ПО для исполнения проекта Специализированное оборудование / ПО для тестирования проекта

◦ В общем случае являются внешними ограничением при решении задачи планирования

18. Роли участников в программных проектах

} Роль – конкретное амплуа сотрудника в конкретном проекте в определенное время

} В программных проектах обычно оперируют ролями, а не сотрудниками

} Виды отношений:

◦ Один сотрудник – одна роль

◦ Один сотрудник – несколько ролей

◦ Несколько сотрудников – одна роль

◦ Несколько сотрудников – несколько ролей

} В общем случае отношение «многие ко многим» } Отношение существует только в контексте одного проекта } Роли часто могут совмещаться } Не все роли присутствуют во всех проектах

} Состав, назначение и функциональные обязанности ролей зависит от конкретного процесса разработки в компании! } В принципе возможно совмещение разных ролей в разных проектах

} Основные роли

◦ Заказчик (customer) ◦ Планировщик ресурсов (planner) ◦ Менеджер проекта (project manager) ◦ Архитектор (architect) ◦ Руководитель команды (team leader, team lead) ◦ Разработчик (developer) ◦ Тестер (tester, QA) ◦ Разработчик документации (technical writer) ◦ Пользователь (user) ◦ Инженер группы поддержки (support engineer)

} Дополнительные

◦ Эксперт предметной области ◦ Специалист по пользовательскому интерфейсу и эргономике ◦ Ответственный за выпуск релизов ◦ Библиотекарь

**ОСНОВНЫЕ РОЛИ**

1. Заказчик

} Инициирует разработку } Участвует в сборе требований } Участвует в разработке спецификации требований } Принимает результаты разработки

1. Планировщик ресурсов

} Член руководства организации } Выдвигает и координирует требования к проектам в организации } Развивает и направляет план выполнения проекта с точки зрения организации } Обеспечивает финансирование проекта

1. Менеджер проекта

} Внешние функции: ◦ Взаимодействие с инициаторам проекта (Заказчиком, Планировщиком ресурсов)

} Внутренние функции: ◦ Распределяет задачи среди членов команды ◦ Организует выполнение проекта ◦ Контролирует процесс разработки

1. Архитектор

} Проектирует архитектуру системы } Разрабатывает основные проектные решения } Определяет общий план развития проекта } Определяет инфраструктуру разработки

1. Руководитель команды

} Является «главным разработчиком» } Осуществляет техническое руководство командой } Разрешает технические вопросы

1. Разработчик

} Реализует проектируемые компоненты } Создает классы и методы } Осуществляет кодирование } Разрабатывает модульные тесты } Выполняет автономное тестирование } Внутри команды может иметь специализацию

1. Тестировщик

} Проверяет качество программного обеспечения (функциональность, надежность, эффективность и т.п.) Составляет тесты для каждой фазы проектирования продукта } Исполняет созданные тесты } Выполняет функциональное тестирование } Выполняет интеграционное, системное тестирование

1. Разработчик документации

} Разработка программной документации } Разработка эксплуатационной документации } Ведение информационной поддержки процесса разработки

1. Пользователь

} Не является заказчиком проекта } Может являться, а может и не являться сотрудником проекта } Является главным потребителем проекта } Обычно существуют группы пользователей проекта

**ДОПОЛНИТЕЛЬНЫЕ РОЛИ**

1. Эксперт предметной области

} Обеспечивает информационную поддержку в предметной области проекта } Если проект большой – таких экспертов может быть несколько

1. Спец по пользовательскому интерфейсу и эргономике

} Проектирует пользовательские интерфейсы } Взаимодействует с заказчиком } Анализирует и оценивает комплексные характеристики интерфейса: ◦ Удобство ◦ Эргономичность ◦ Лаконичность ◦ Дружественность ◦ Локализуемость

**Эргономичность** — наличие условий, возможностей для лёгкого, приятного, необременительного пользования чем-либо или удовлетворения каких-либо нужд, потребностей

1. Ответственный за выпуск релиза

} Определяет и реализует политику выпуска релизов } Формулирует и проверяет требования к конкретному релизу: ◦ Необходимая функциональность ◦ Состав релиза } Определяет дату выхода релиза } Контролирует процесс выхода релиза

1. Библиотекарь

} Ведет библиотеку проекта } Контролирует соответствие выпускаемого продукта принятым стандартам

19. Проектные активности программных проектов

} Проект – самостоятельно управляемый элемент разработки } Нормальный результат программного проекта – программный продукт

} Проектные активности ◦ Выполнение задач из ТЗ (подчиненных проектов, работ) ◦ Реализация изменений ◦ Исправление дефектов } Временные сущности: ◦ Этапы (stage) ◦ Вехи (milestone)

} Задача – часть программного проекта, обладающая следующими свойствами:

◦ С задачей связан определенный набор требований

◦ Задача может реализовываться относительно самостоятельно

◦ Результат выполнения задачи можно проконтролировать

**Атрибуты задачи**

} Планируемое время старта задачи } Планируемое время завершения задачи } Список вложенных подзадач } Задача, которой подчинена данная } Список ресурсов, требующихся для реализации данной задачи } Временные условия старта задачи } Временные события, инициируемые завершением задачи

} Временные связи между задачами:

◦ Последовательное выполнение A начинается после окончания B A начинается не раньше, чем через t после окончания B A начинается не позже, чем через T после окончания B A начинается не раньше, чем через t и не позже, чем через T после окончания B

◦ Параллельное выполнение A начинается вместе с B A начинается не раньше, чем через t после старта B A начинается не позже, чем через T после старта B A начинается не раньше, чем через t и не позже, чем через T после старта B

} Не должно быть циклических связей

**Изменения проекта**

} Изменение проекта вызваны одобренными изменениями требований } Для программного проекта изменение – вид работы } Все изменения вносятся в проектный план

**Исправление программных дефектов**

} Программный дефект (bug) – обнаруженные в процессе тестирования или наблюдения: ◦ Программная ошибка ◦ Несоответствие спецификации ◦ Несоответствие стандарту ◦ …

} Для программного проекта исправление дефекта – также один из видов работы

20. Временные сущности программных проектов

**Этап программного проекта**

} Этап проекта – множество задач проекта, подчиненных достижению какой-либо локальной цели. } Обычно этап – элемент проекта, видимый заказчику } К этапам обычно привязано финансирование проекта } Завершение этапа может сопровождаться ◦ Созданием макета ◦ Выпуском версии продукта ◦ Реализации компонента продукта ◦ И т.п.

} По окончании этапа можно принимать кардинальные решения: ◦ Продолжение проекта ◦ Прекращение проекта ◦ Перепланирование ◦ Изменение финансирования

**Вехи проекта**

} Веха – milestone } Веха – законченная часть какого-либо этапа работы } Достижение вехи можно наблюдать и контролировать } Вехи – те контрольные точки, по которым можно грубо оценить успешность всего проекта } В зависимости от способа организации проекта веха может быть: ◦ Видимой только менеджеру ◦ Видимой менеджеру и заказчику

**Выполнение проекта**

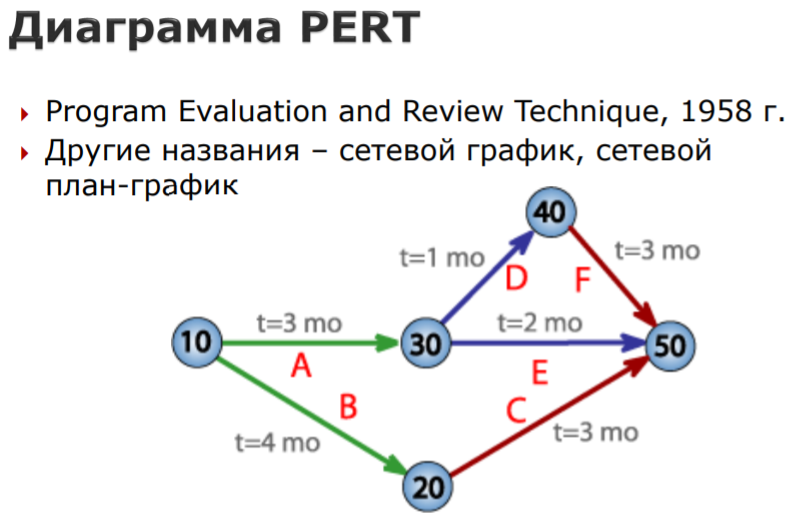
} Процесс выполнения программного проекта – взаимосвязанное существование во времени: ◦ Проектных активностей ◦ Ресурсов ◦ Временных сущностей

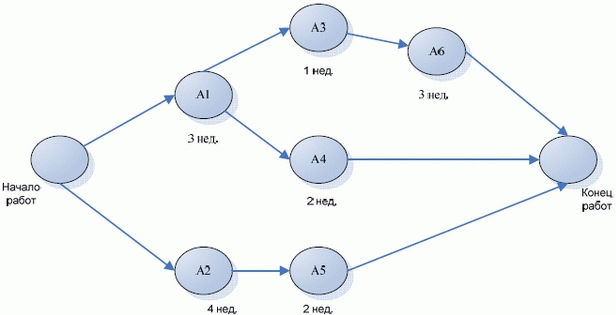
21. Визуализация плана. Наблюдение за программным проектом

} Визуализация – основной способ планирования, контроля и наблюдения } Для классических (не agile) методологий существует два основных подхода: ◦ Диаграммы Ганта ◦ Диаграммы PERT

****

По сути, диаграмма Ганта состоит из полос, ориентированных вдоль оси времени. Каждая полоса на диаграмме представляет отдельную задачу в составе проекта (вид работы), её концы — моменты начала и завершения работы, её протяженность — длительность работы. Вертикальной осью диаграммы служит перечень задач. Кроме того, на диаграмме могут быть отмечены совокупные задачи, проценты завершения, указатели последовательности и зависимости работ, метки ключевых моментов (вехи), метка текущего момента времени «Сегодня» и др.

****



Когда стрелка рисуется от задачи А к задаче B, то это означает, что задача А должна быть обязательно завершена до начала задачи B. Вы можете создать параллельные задачи, которые могут быть расположены на той же стадии, но на разных линиях задач. Они не должны зависеть друг от друга.

**Наблюдение за проектом ведется по**

} Активности: ◦ Задачи, дефекты, фиксации изменений

} Ресурсам: ◦ Сотрудники, время

} Временным сущностям: ◦ Этапы, вехи, критический путь

**Виды срезов**

◦ Задачи ◦ Сотрудники ◦ Вехи ◦ Дефекты ◦ Фиксации изменений (коммиты) в СКВ ◦ Критический путь

1. Срез по задачам

◦ Сотрудники, занятые решением задачи ◦ Соответствие задач – графикам ◦ Процент завершенности по задачам проекта ◦ Общее количество дефектов задачи ◦ Количество незакрытых дефектов задачи

1. Срез по сотрудникам

◦ Текущие задачи сотрудника ◦ Отставание от графика сотрудника ◦ Общее количество дефектов, относящихся к сотруднику ◦ Количество незакрытых дефектов, относящихся к сотруднику

1. Срез по дефектам

◦ Количество дефектов для каждой задачи ◦ Количество незакрытых дефектов для каждой задачи ◦ История изменения дефектов ◦ Среднее время исправления дефекта ◦ Среднее количество дефектов у сотрудников ◦ Распределение дефектов по сотрудникам

1. Срез по фиксациям изменений (коммитам) В СКВ

◦ Среднее число фиксаций на сотрудника за единицу времени ◦ Равномерность фиксаций у сотрудников

1. Срез по критическому пути

◦ Сотрудники в критическом пути ◦ Задачи в критическом пути ◦ Временные запасы в критическом пути

Критический путь– это самая длинная по срокам последовательная цепочка работ.

22. Понятие дефекта программного обеспечения. Характеристики дефектов

Системы отслеживания ошибок

} Другие названия:

◦ Системы управления дефектами ◦ Системы управления изменениями и дефектами ◦ Трассировка изменений ◦ Трассировка ошибок ◦ Управление инцидентами ◦ Управление рекламациями ◦ Bug tracking system, issue tracking system

} Дефект – обнаруженная в процессе разработки, тестирования или эксплуатации ошибка в разрабатываемом приложении } Исправление обнаруженной ошибки – самостоятельная проектная активность

**Характеристики дефектов**

} Идентификатор дефекта } Состояние дефекта } Описание дефекта } Автор } Время обнаружения } Контекст } Срочность } Серьезность (важность) } Категория } Ответственный за исправление } Ответственный за проверку } Версия, в которой дефект исправлен } Зависимости } Временные параметры устранения дефекта } Дополнительные параметры: ◦ Резолюция ◦ Способы обхода ◦ URL ◦ Attachments

1. Идентификатор дефекта

} Уникальный идентификатор, выданный дефекту } Не должен повторяться ни в одном из проектов предприятия } Должен быть легкий механизм поиска дефекта по идентификатору

} Не должен изменяться в процессе жизненного цикла проекта, т.к.: ◦ Могут быть ссылки в документации ◦ Могут быть ссылки от внешних заинтересованных сторон ◦ Могут быть ссылки по зависимостям

1. Контекст дефекта (с каким проектом и задачей связан)

} Дефект обычно связан с каким-либо проектом или задачей } Должна указываться версия (версии) проекта или задачи. Это может быть ◦ Внешняя версия проекта ◦ Ветка разработки } В процессе жизненного цикла проект и версия могут уточняться

1. Содержание дефекта (описание)

} Текстовое описание, дающее исчерпывающее представление о проявлении дефекта } По этому описанию должно быть возможно повторить условия, в которых проявляется дефект } Может содержать ссылки на требования и т.п.

1. Срочность дефекта

} Приоритет дефекта } Показывает относительную срочность исправления дефекта с точки зрения нашедшего его } Обычно выражается в относительных единицах: ◦ Низкая ◦ Средняя ◦ Высокая ◦ Срочная

1. Категория дефекта (тип)

} Описывает тип найденного дефекта } Возможные категории: ◦ Функциональный дефект ◦ Дефект документации ◦ Дефект требований ◦ Предложение по усовершенствованию \* ◦ Новая функция \*

1. Серьезность дефекта (влияние на проект)

} Показывает степень влияния проявления дефекта на проект } Возможные варианты ◦ Косметический дефект ◦ Рабочий дефект ◦ Дефект, вызывающий зависание приложения ◦ Дефект, вызывающий аварию приложения ◦ Дефект, вызывающий потерю или нарушение целостности данных ◦ И т.п.

1. Автор дефекта (тот кто обнаружил)

} Автор – лицо (сотрудник), обнаружившее дефект } Автором может быть ◦ Тестер ◦ Заказчик ◦ Пользователь ◦ Разработчик Собственно автор кода Другой разработчик И т.п. } Автором может быть любой, имеющий права фиксировать дефекты для данного проекта

1. Ответственный за исправление дефекта

} Ответственный за исправление дефекта – лицо, в задачу которого входит устранение дефекта } В зависимости от политики управления ответственный за исправление дефекта ◦ Может назначаться автоматически (например, менеджер проекта) ◦ Может явно назначаться вручную

1. Ответственный за проверку дефекта

} Ответственный за проверку дефекта – лицо, в задачу которого входит проверка успешности устранения дефекта } Ответственный за проверку не обязательно автор дефекта! } В зависимости от политики управления ответственный за проверку дефекта ◦ Может назначаться автоматически (например, тестер проекта) ◦ Может назначаться вручную

1. Состояние дефекта

} Показывает этап жизненного цикла дефекта } Возможные состояния ◦ Новый ◦ Взятый на исправление ◦ Исправленный ◦ Закрытый (исправленный и проверенный) ◦ Незакрытый (исправленный, но проваливший проверку) ◦ Отклоненный ◦ Дубликат ◦ Заново открытый ◦ Временно приостановленный ◦ Требующий пояснения ◦ Не проявляющийся

1. Зависимости дефекта (связь между исправлением дефектов)

} Показывает зависимости исправления данного дефекта от исправления других дефектов } Зависимости представляются в виде списка идентификаторов дефектов, от которых зависит данный дефект

1. Временные параметры устранения дефекта

} Желаемое время, когда требуется устранить дефект } Желаемое версия проекта, к которой требуется устранить дефект

1. Резолюция на дефект (решение ответственного лица на тему дефекта)

} Необязательная текстовая реакция ответственного за исправление } Может сопровождать переход дефекта из одного состояния в другое

1. Способы обхода дефекта

} Необязательная текстовая реакция ответственного за исправление } Показывает, как можно использовать систему до окончательного исправления дефекта } Может сопровождать переход дефекта из одного состояния в другое

23. Жизненный цикл дефекта.

} Изменение характеристик дефекта

◦ Переход из одного состояния в другое состояние (статус выполнения)

◦ Изменение ответственного за исправление

◦ Изменение ответственного за проверку

◦ РЕДКО: изменение автора

◦ Изменение контекста (с каким проектом и задачей связан)

◦ РЕДКО: Изменение серьезности

◦ Изменение срочности

◦ РЕДКО: Изменение категории (тип – документация, функция и тд)

◦ Изменение содержания (описание как проявляется)

◦ Изменение резолюции (необязательная текстовая реакция ответственного за исправление)

◦ Изменение способа обхода

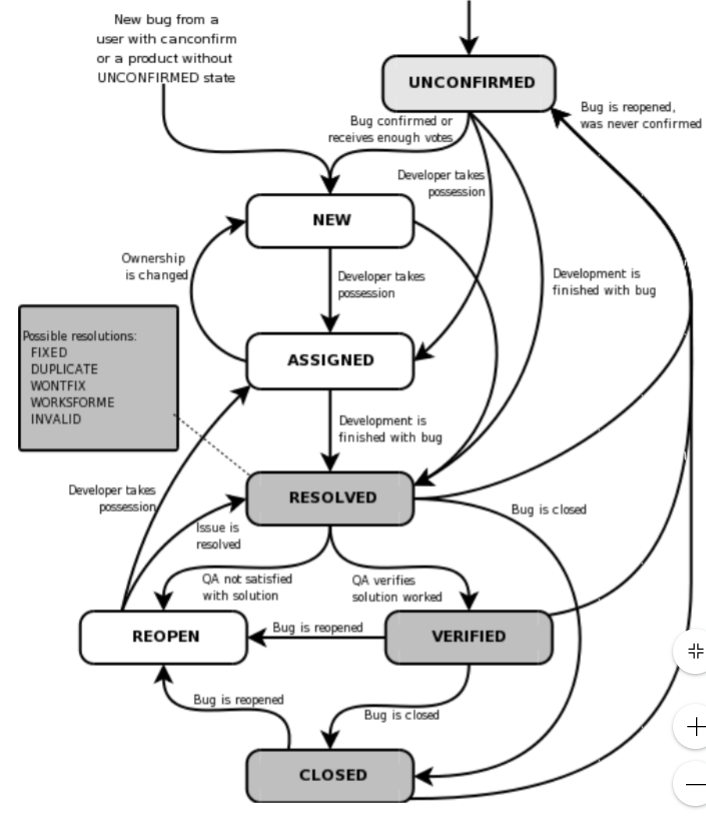
} Права на изменения отдельных характеристик и отдельные переходы состояний зависят от ◦ Состояния дефекта ◦ Роли сотрудника ◦ Политики предприятия

24. Системы управления дефектами.

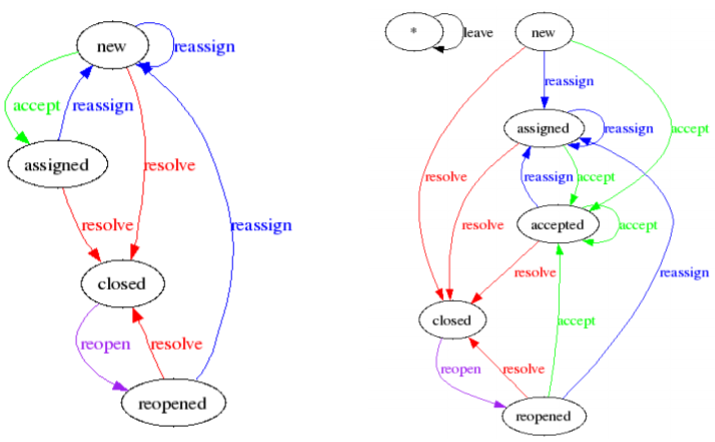
} Коммерческие ◦ IBM Rational ClearQuest ◦ Microfocus StarTeam ◦ Atlassian JIRA ◦ JetBrains YouTrack ◦ …

} Свободно распространяемые ◦ Mozilla Bugzilla ◦ Trac ◦ Redmine ◦ MantisBT ◦ TUTOS ◦ Интегрированные в серверы хостинга проектов (BitBucket, GitHub и т.п.) ◦ … (более 50 средств)

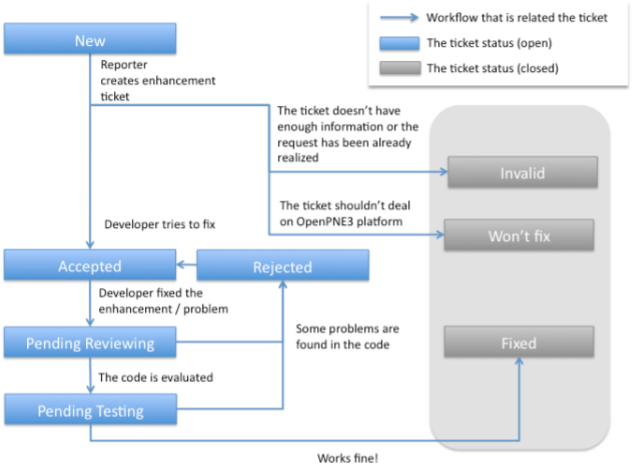
1. Mozilla Bugzilla



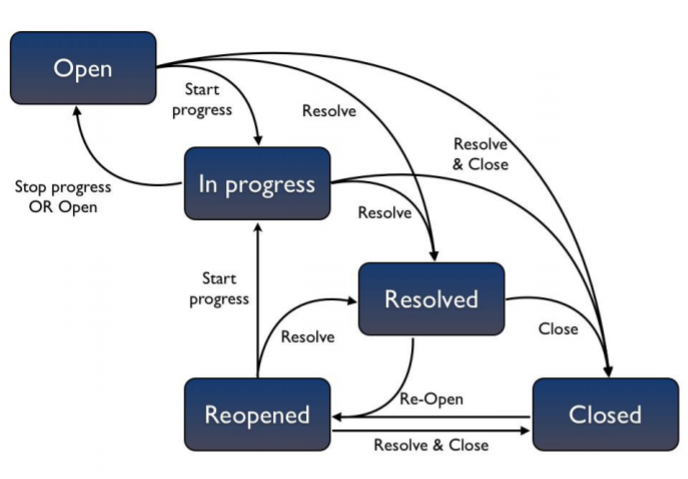
1. Trac



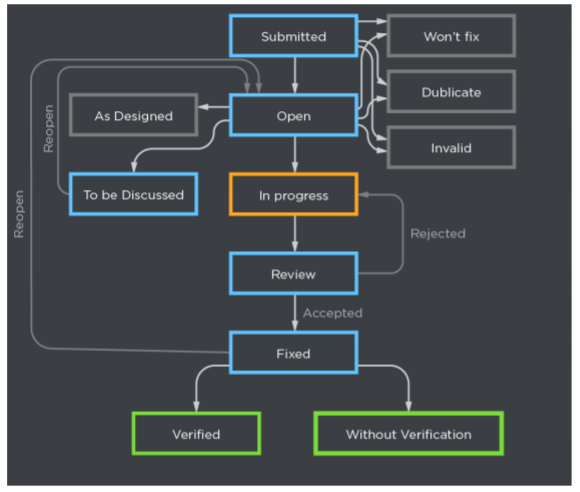
1. Redmine



1. Atlassian Jira

****

1. JetBrains YouTrack



**Нотификации в системах управления дефектами**

} Причины нотификации ◦ Появление нового дефекта ◦ Изменение состояния существующего дефекта ◦ Изменение характеристик существующего дефекта

} Способы нотификации ◦ Интерактивные SMS Системы обмена мгновенными сообщениями ◦ Не интерактивные E-mail Через клиент СУД

} Политика нотификации ◦ По отношению к роли ◦ По отношению к свойствам дефекта ◦ По отношению к состоянию дефекта

} В развитых СУД можно гибко настраивать способы, условия и задействованные лица нотификации

25.Управление риском в программных проектах: идентификация, анализ, ранжирование

} Риск – возможность неудачи, неудовлетворительного результата } Риски в программных проектах:

◦ Бюджет Превышение бюджета

◦ Сроки Превышение сроков

◦ Функциональность Низкая надежность Некорректное функционирование Низкое качество

} R = P(UR)∙L(UR) ◦ R – показатель риска ◦ P – вероятность неуспешного результата ◦ L – потери от неуспешного результата

} Идентификация риска } Анализ риска } Ранжирование риска } Планирование управления риском } Разрешение риска } Наблюдение за риском

****

1. Идентификация рисков

Формирование списка элементов риска данного проекта } Три категории рисков: ◦ Проектные риски ◦ Технические риски ◦ Коммерческие риски

* 1. Проектные риски

} Выбор бюджета, плана, человеческих ресурсов проекта } Формирование требований к продукту } Проблемы с кадрами } Сложность, размер и структура программного проекта } Методика взаимодействия с заказчиком

* 1. Технические риски

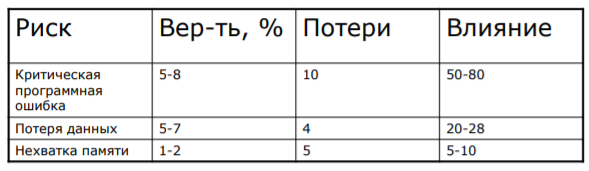
} Трудности этапов проектирования, реализации, тестирования и сопровождения } Неполнота или неточность спецификаций } Сомнительность принятых технических решений

* 1. Коммерческие риски

} Продукт не требуется на рынке } Продукт слишком устарел } Продукт слишком новаторский } Возможность прекращения финансирования

1. Анализ риска

Оценка вероятности возникновения каждого типа рисков и величины потерь } Вероятности определяются на основе экспертных оценок и статистики } Все риски заносятся в таблицу:



1. Ранжирование рисков

} Сортировка рисков, пропорционально влиянию } 20% элементов риска – обычно составляют 80% суммарного проектного риска

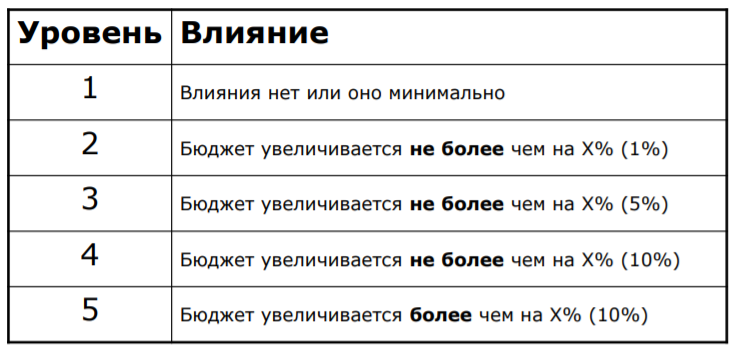
26.Управление риском в программных проектах: планирование, разрешение, наблюдение.

4) Планирование управления риском

} Цель – сформировать набор функций управления каждым элементом риска } Выбираются эталонные уровни риска – такие которые могут быть причиной прекращения проекта

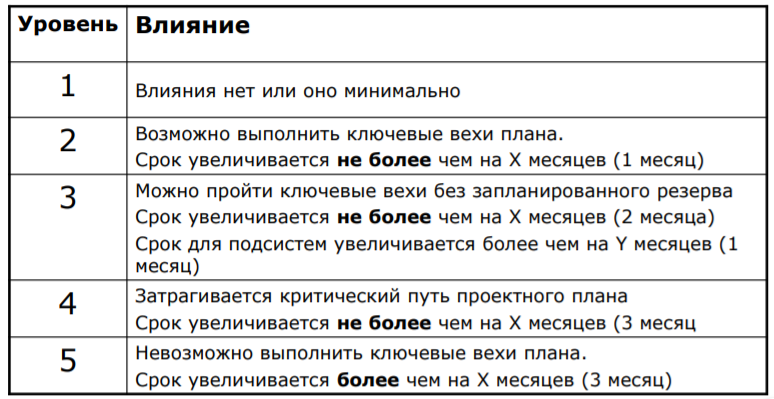
4.1) Превышение стоимости

Уровни влияния на стоимость



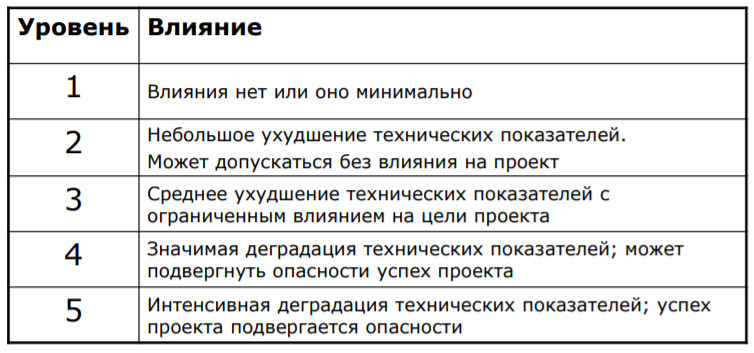
4.2) Срыв планирования

Уровни влияния на сроки



4.3) Деградация технических показателей (характеристик)

Уровни влияния на технические показатели



****

} Строятся зависимости между элементом риска и эталонными уровнями риска } Строится план управления каждым элементом риска } План интегрируется в общий план проекта

5-6) Разрешение и наблюдение риска

} Разрешение – плановое применение действий по уменьшению риска

} Наблюдение ◦ Цикличность ◦ Корректировка

**Методика управления рисками номер 2**

} Выполняется выделение и ранжирование наиболее существенных элементов риска

} Производится планирование регулярных просмотров процесса разработки ◦ В больших проектных группах – ежемесячно ◦ В остальных проектах – чаще

} При просмотре рассматриваются 10 верхних элементов риска ◦ Фиксируется приоритет каждого из 10 рисков ◦ Фиксируется предыдущий приоритет ◦ Частота попадания в верхнюю часть списка ◦ Обсуждается прогресс в разрешении риска

27.Предпосылки для версионирования ПО. Ветвление.

1. Групповая работа с проектом

} Основные задачи: ◦ Повышение надежности хранения артефактов ◦ Общий доступ к файлам ◦ Сохранение истории модификации файла ◦ Возможность возврата к предыдущим версиям ◦ Пометка отдельных версий файла ◦ Поддержание и развитие нескольких параллельных историй артефакта ◦ Поддержка нескольких конфигураций проекта ◦ Сравнение версий ◦ Объединение разрозненного кода

1. Общий доступ к файлам

} Одновременное редактирование одного файла разными пользователями } Потеря изменений, сделанных пользователем (затерты записью изменений другого пользователя) \* В параллельных системах для разрешения используются семафоры, мьютексы, критические секции и т.п.

1. Сохранение истории модификации файла

} Необходимо иметь историю изменения файла: ◦ Версия ◦ Автор изменения ◦ Время изменения ◦ Суть изменения ◦ Причина изменения

1. Пометка отдельных версий файла

} Отдельные версии файлов необходимо специально помечать } Причина пометки: ◦ Качественная версия ◦ Версия, обладающая определенными свойствами ◦ Версия, являющаяся частью релиза проекта определенной версии

1. Поддержание и развитие нескольких историй файла

} Причины разветвления версий файла:

◦ Развитие нескольких версий проекта Поставленных заказчику Разрабатываемых

◦ Разработка новой функциональности

1. Поддержка нескольких конфигураций проекта

} Наличие нескольких конфигураций проекта ◦ Для разной аппаратуры ◦ Для разных операционных систем ◦ Для разных комплектов поставки } Разработка новой (экспериментальной) функциональности

**Ветви файлов**

} Ветвь (ветка, branch) – механизм, который служит для ветвления дерева ревизий файла } Имя ветви однозначно определяет группу ревизий (ветвь) } Имя ветви используется для переключения между ветвями ревизий файла

28.Системы контроля версий. Типы СКВ. Общие принципы организации.

} СКВ предназначены для автоматизации групповой работы и управления версионированием проектов } СКВ обеспечивают ◦ Репозиторий (или репозитории) хранения проектов ◦ Стандартные операции для групповой работы ◦ Клиенты для выполнения операций

**Типы СКВ**

} Централизованные файл-серверные СКВ

◦ Единое централизованное хранилище ◦ Файл-серверный доступ ◦ Примеры: MS Visual Source Safe

} Централизованные клиент-серверные СКВ

◦ Единое централизованное хранилище ◦ Клиент-серверный доступ ◦ Примеры: CVS, Subversion

} Распределённые СКВ ◦ Репозиторий хранится на каждом компьютере ◦ Сетевая синхронизация репозиториев посредством слияний (заплаток, патчей, change sets и т.п.) ◦ Используется в интернет-проектах, когда разработчики существенно удалены друг от друга ◦ Примеры: mercurial, git, Bazaar и т.п.

**Ревизия файла**

} Ревизия файла – уникальный идентификатор версии файла в системе контроля версии ◦ CVS: 1.2 ◦ SVN: 238 ◦ Git, Mercurial: хэш SHA-1

} При изменении файла номер ревизии изменяется определенному правилу ◦ иногда номер ревизия является атрибутом всего репозитория

} Атрибуты ревизии ◦ Идентификатор ◦ Автор изменения ◦ Дата ◦ Текстовое описание изменения ◦ Внешние атрибуты Тэги Идентификаторы ветвей

**Хранимые копии проекта**

} Для централизованных СКВ: ◦ Локальная копия проекта ◦ Локальная копия проекта, находящегося под контролем СКВ ◦ Серверная копия, находящая в репозитории

} Для распределенных СКВ: ◦ Локальная копия проекта ◦ Локальная копия проекта, находящегося под контролем СКВ ◦ Копия, находящая в локальном репозитории ◦ Копия, находящая в удаленном репозитории

**Общие принципы хранения файлов в системах версионирования**

} Поддержка текстового и бинарного формата хранения } Для текстового формата: ◦ Хранение инкрементных изменений ◦ Возможность визуального сравнения ревизий } Для бинарного формата: ◦ Хранение полных версий артефактов

**Пометка версий в СКВ**

1. Тэги

} Тэг – текстовая метка, привязанная к какой-либо ревизии файла или репозитория } Одна ревизия может содержать несколько тэгов } Выборку ревизии файла/файлов можно производить по тэгам

1. Виртуальные каталоги

} Вместо тэга создается каталог } В него помещаются виртуальные копии необходимых ревизий всех требуемых файлов } Работа с таким каталогом происходит стандартными способами

29.Системы контроля версий. Типовые операции

} Импорт проекта (import) } Экспорт проекта (export) } Получение проекта (checkout, clone, …) } Обновление файла (update, fetch, ..) } Фиксация изменений (commit) } Сравнение изменений } Установка тэгов } Переход к другой ревизии (откат) } Создание ветвей (fork,…) } Переключение на ветвь } Слияние (merge) } Разрешение конфликтов } Блокировка файлов } Выгрузка изменений (push) } Запрос на изменение (pull request)

1. Импорт проекта - Первоначальное помещение локального проекта в репозиторий
2. Экспорт проекта • Извлечение проекта из СКВ в локальный каталог • Удаление проекта из СКВ
3. Получение проекта (clone): } Получение локального слепка проекта } Получение осуществляется по одному из критериев: ◦ Головная версия (HEAD, trunk, default, master, …) ◦ Версия на определенную дату ◦ Версия с определенным тэгом ◦ Версия из определенной ветви
4. Фиксация изменения (commit) • Посылка измененной версии файла в репозиторий • Операция игнорируется, если ревизия на сервере изменилась
5. Обновления файлов • Копирование свежей версии из репозитория • Слияние локальных изменений и серверных в локальном файле
6. Разрешение конфликтов: } Конфликт – когда одна и та же строка была поразному изменена в двух версиях } Если при слиянии произошел конфликт – в текст попадают обе версии участков кода с пометками } Разрешение проводится только в локальной копии } В репозитории хранятся только утвержденные версии с разрешенным конфликтом \* Отсутствие конфликта не означает отсутствия ошибок!
7. Сравнение изменений ◦ Действует только для текстовых файлов ◦ Сравнивать можно любые две ревизии одного файла из любых ветвей проекта

30.Сборка программных проектов. Проблемы при сборке программных проектов

} Сборка (building) программного проекта– набор правил и процедур, направленный на получение исполняемой программы

} Выпуск (release) программного продукта – процесс отчуждения программы от разработчика и заключающийся в ◦ Сборке программного проекта ◦ Формировании инсталляционного пакета ◦ Формировании документации ◦ Формировании аннотации релиза

**Задачи сборки и выпуска проектов:**

} Трансляция всего проекта } Сборка дистрибутива } Подготовка исходных текстов } Подготовка документации

**Причины сборки проекта**

} Проверка работоспособности } Очередная периодическая сборка (еженощная) } Подготовка версии к автоматическому тестированию } Подготовка дистрибутива } Инсталляция дистрибутива (из исходных текстов)

**Процесс сборки**

В сборку входят: исходные файлы (текста), подключаемые файлы, библиотеки, процедуры сборки проекта и его компонентов, средства сборки.

**Проблемы сборки проектов**

1. Исходные тексты

} Исходными для сборки являются ◦ Исходные файлы ЯП ◦ Шаблоны ◦ Конфигурационные файлы ◦ Ресурсные файлы ◦ Файлы документации ◦ И т.п.

} Проблемы с исходными текстами ◦ Отсутствие всех исходных текстов ◦ Некорректное расположение исходных текстов ◦ Неправильные версии исходных файлов

1. Подключаемые файлы

} Подключаемые файлы ◦ Сторонние файлы, необходимые для сборки проекта. Обычно содержат интерфейсы к библиотечным функциям и основные типы данных

} Проблемы с подключаемыми файлами ◦ Отсутствие подключаемых файлов ◦ Неправильное расположение подключаемых файлов ◦ Некорректные версии подключаемых файлов

1. Библиотеки

} Библиотеки ◦ Некоторые компиляторы требуют наличия не только статических, но динамических библиотек с ранним связыванием

} Проблемы с используемыми библиотеками ◦ Отсутствие библиотек ◦ Неправильное расположение библиотек ◦ Некорректные версии библиотек

1. Процедуры сборки

} Проблемы с процедурами сборки ◦ Отсутствие процедуры сборки проекта ◦ Отсутствие процедур сборки компонентов проекта ◦ Несоответствие версии процедуры сборки

1. Системное окружение

} Проблемы с системной средой и аппаратной платформой ◦ Неправильная версия ОС ◦ Неправильные версии компонентов ОС

◦ Некорректный состав аппаратуры Тип процессора Количество ядер И т.п.

◦ Некорректные параметры аппаратуры Скорость процессора Объем памяти Объем дисков И т.п.

1. Средства сборки

} Утилиты сборки: ◦ Трансляторы (компиляторы, ассемблеры и т.п.) ◦ Редакторы связей (линкеры) ◦ Библиотекари ◦ Архиваторы ◦ Генераторы дистрибутивов ◦ Генераторы документации ◦ Генераторы кода ◦ И т.п.

} Проблемы со средствами (утилитами) сборки ◦ Неполный состав средств сборки ◦ Некорректные версии средств сборки

31.Сборка программных проектов. Окружение для сборки. Общие требования к системе сборки. Версии в программных проектах

**Окружение для сборки проетка**

} Аппаратная платформа } Системное окружение ◦ Операционная система ◦ Системные файлы } Библиотечное окружение ◦ Подключаемые файлы ◦ Библиотечные файлы } Исходные файлы в требуемых каталогах } Средства сборки

} Построение окружения ◦ Установка всех требуемых библиотек Из дистрибутивов (версия разработчика) ◦ Установка всех средств сборки Из дистрибутива ◦ Размещение всех исходных файлов Из СКВ Из архива

**Общие требования к сборке проекта**

} Сборка должна проводиться на любом компьютере с подготовленным окружением

} Сборка должна проводиться отдельно от рабочего места разработчика ◦ Версии подключаемых файлов ◦ Версии библиотек ◦ Расположение исходных файлов ◦ Версии средств сборки ◦ Влияние временных файлов Объектные файлы Исполняемые файлы Предкомпилированные заголовки

} Процедура сборки должна быть: ◦ Документирована ◦ Прозрачна ◦ Повторяема

**Способы сборки программных проектов**

} С помощью среды разработки } С помощью запуска компилятора командной строки } С помощью сценариев } С помощью различных средств сборки ◦ make ◦ nmake ◦ Apache Ant ◦ Apache Maven ◦ Gradle ◦ И т.п.

**Сборка компонента проекта**

} Для каждого компонента проекта должна быть сформирована локальная процедура сборки ◦ Например, makefile } В каждом компоненте проекта сборка должна проходить одной командой ◦ Например, make

**Требования к процедурам сборки**

} Процедуры сборки должны ◦ находиться под управлением СКВ ◦ помечаться и ветвиться аналогично файлам проекта ◦ изменяться в соответствии с изменениями проекта

**Средства сборки**

} Коммерческие средства управления сборкой: ◦ IBM Rational Build Forge ◦ FinalBuilder ◦ TeamCity ◦ Visual Build ◦ Anthill Pro } Бесплатные средства управления сборкой : ◦ GNU Make ◦ Nmake ◦ Ant ◦ Maven ◦ Gradle } … (около 100)

32.Непрерывная интеграция.

Интеграция ПО – процесс объединения частей ПО (вновь добавленных или измененных) в основное ПО с целью получения работоспособной версии

Непрерывная интеграция (continuous integration или CI) — один из процессов ПИ, предполагающий периодическую (частую) интеграцию отдельных частей проекта

**Требования к проектам**

• Все компоненты проекта находятся в СКВ • Операция сборки проекта — автоматизирована • Операция тестирования проекта — автоматизирована

**Действия в системах непрерывной интеграции**

• Инкремент текущего номера сборки («билда») • Пометка текущим номером сборки файлов собираемого проекта • Получение проекта (помеченного тэгом) из репозитория • Сборка проекта • Запуск тестирования (и/или других процедур обеспечения качества) • Развертывание проекта • Формирование отчета

**Условия запуска сборки**

1. По расписанию

• Обычно запуск производится ночью (nightly build, daily build) • Проводится запуск полного набора тестов

1. По обновлению

• Цель — определить, что данное обновление (фиксация) не испортило систему • Запуск производится после каждого обновления в репозитории проекта • Запускается сокращенная процедура сборки и тестирования • Длительность 5-15 минут • При большей длительности — интеграция бессмысленна

1. По запросу

**Промышленные средства непрерывной интеграции**

• Apache Gump • Atlassian Bamboo • CruiseControl • Jenkins • BuildBot • Travis CI • MS Team Foundation Server • JetBrains TeamCity • … (более 20 средств)

33.Непрерывная доставка. Непрерывное развертывание.

} CI/CD ◦ Continuous Integration/ Continuous Delivery ◦ Continuous Integration/ Continuous Deployment

1. **Непрерывная доставка**

1.1) Поучение ПО из репозитория } Автоматическая сборка ПО } Запуск автоматических тестов

1.2) Развертывание на тестовом сервере } Дополнительное тестирование (автоматическое и силами QA) } Размещение обновления

1.3) **Ручное** принятие решения о развертывании обновления на продуктовом сервере } Автоматическое развертывание на продуктовом сервере

**2) Непрерывное развертывание**

1.1) Поучение ПО из репозитория } Автоматическая сборка ПО } Запуск автоматических тестов

1.2) Развертывание на тестовом сервере } Дополнительное тестирование (автоматическое и силами QA) } Размещение обновления

1.3) Автоматическое развертывание на продуктовом сервере

Доставка отличается от развертывания тем, что в 1 случае мы вручную определяем решение о развертывании обновления на продуктовом сервере, а во 2 – автоматически

34.Документирование ПО. Виды программной документации.

} Виды документации ◦ Проектная документация ◦ Программная документация ◦ Эксплуатационная документация ◦ Рабочая документация (\*)

1. Проектная документация

} Должна создаваться в начале проекта и модифицироваться по мере продвижения проекта

} Стандарты: ◦ ГОСТ 19.201-78 ЕСПД. Техническое задание. Требования к содержанию и оформлению. ◦ ГОСТ 34.602-89 Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы ◦ ГОСТ 19.301-79 ЕСПД. Программа и методика испытаний. Требования к содержанию и оформлению.

1. Программная документация

} Должна создаваться одновременно с реализацией проекта и модифицироваться в процессе эволюции проекта

} Стандарты: ◦ ГОСТ 19.202-78 ЕСПД. Спецификация. Требования к содержанию и оформлению. ◦ ГОСТ 19.401-78 ЕСПД. Текст программы Требования к содержанию и оформлению ◦ ГОСТ 19.402-78 ЕСПД. Описание программы. ◦ ГОСТ 19.504-79 ЕСПД. Руководство программиста. Требования к содержанию и оформлению ◦ ГОСТ 19.701-90 (ИСО 5807-85) ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. (Вместо ГОСТ 19.002-80, ГОСТ 19.003-80)

1. Эксплуатационная документация

} Должна создаваться по окончании проекта

} Стандарты: ◦ ГОСТ 19.505-79 ЕСПД. Руководство оператора. Требования к содержанию и оформлению. ◦ ГОСТ 19.508-79 ЕСПД. Руководство по техническому обслуживанию. Требования к содержанию и оформлению. ◦ ГОСТ 19.503-79 ЕСПД. Руководство системного программиста. Требования к содержанию и оформлению

1. Рабочая документация

} Документация, отражающая текущее состояние проекта } Рабочая документация подчиняется только корпоративным стандартам документирования } Рабочая документация должна изменяться вместе с проектом } Рабочая документация должна версионироваться вместе с проектом } На стадии выпуска рабочая документация может стать программной документацией или эксплуатационной документацией

} Обычной практикой является электронное ведение рабочей документации} Вариант ведения электронной документации - портал проекта ◦ Разделяемое дисковое пространство ◦ Группы новостей ◦ Списки рассылки ◦ Wiki ◦ Коммерческие системы организации совместной работы MS Sharepoint IBM Lotus И т.п.

Бывает:

} Текстовая документация } Графическая документация ◦ Схемы программ ◦ Схемы алгоритмов ◦ Схемы данных ◦ UML-диаграммы ◦ И т.п. } Динамическая документация (\*)

35.Документирование ПО. Использование UML для документирования

} UML является: ◦ Средством описания архитектуры и особенностей реализации информационных систем ◦ Средством общения заказчика и проектировщика ◦ Формальной моделью (артефактом) какой-либо стадии проектирования ◦ Способом документирования информационной системы

**UML** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Unified Modeling Language* — унифицированный язык моделирования) — язык [графического](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B7%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) описания для [объектного моделирования](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE%D0%B5_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5&action=edit&redlink=1) в области [разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), для [моделирования](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [бизнес-процессов](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81), [системного проектирования](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и отображения [организационных структур](https://ru.wikipedia.org/wiki/%D0%9E%D1%80%D0%B3%D0%B0%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0).

UML является языком широкого профиля, это — [открытый стандарт](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82), использующий графические обозначения для создания [абстрактной модели](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) [системы](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), называемой *UML-моделью*. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна [генерация кода](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4%D0%BE%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F).

**Использование UML в документировании**

Используются на всех стадиях проектирования:

◦ Сбор и разработка требований Диаграмма случаев использования Диаграммы деятельностей

◦ Разработка архитектуры Диаграммы классов Диаграммы последовательностей Диаграммы взаимодействия Диаграммы деятельностей Диаграммы пакетов

◦ Реализация Диаграммы классов Диаграммы объектов Диаграммы состояний Диаграммы последовательностей Диаграммы взаимодействия Диаграммы пакетов

◦ Тестирование Диаграмма случаев использования

◦ Сопровождение Диаграмма компонентов Диаграмма развертывания

36.Документирование ПО. Генерация документации

} Впервые предложил Д. Кнут – язык WEB ◦ Концепция «грамотного» программирования ◦ Объединения текста и документации ◦ Один компилятор для получения объектного кода ◦ Второй компилятор для получения документации

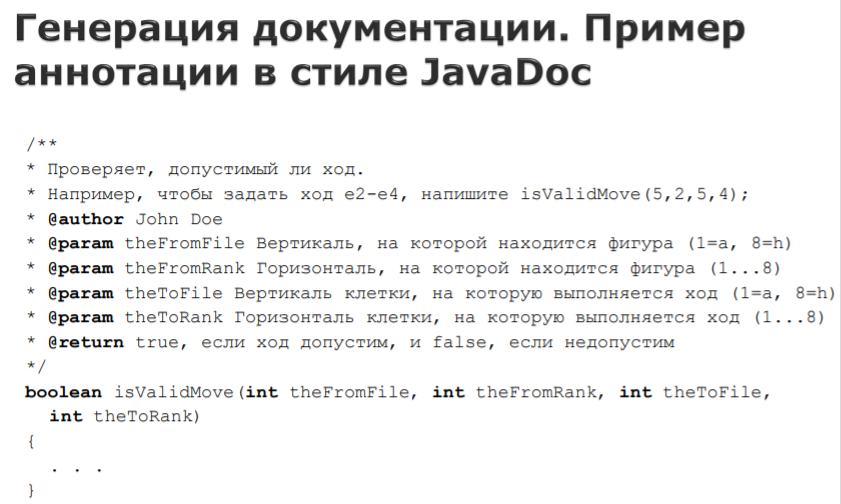
} Сейчас основной подход – аннотирование текста программы с помощью структурированных комментариев ◦ Описание модулей ◦ Описание функций, классов ◦ Описание переменных ◦ Описание параметров функций и методов ◦ И т.п.

} На основании аннотаций и текста программы генерируется система документов: ◦ html – страницы ◦ Документ Word, rtf ◦ Документ PDF ◦ Документы XML

} Используется при подготовке программной документации ◦ Описание программы ◦ Руководство программиста

} Генераторы документации

◦ Java - Javadoc



◦ C++ - Cppdoc

◦ Object Pascal - Pasdoc

◦ JavaScript - JSDoc

◦ Многоязыковые: Doxygen

37.Документирование ПО. Промышленные технологии документирования.

} Основные принципы

◦ Принцип единого источника ◦ Использование структурированных текстовых форматов хранения документации ◦ Отделение семантики от форматирования ◦ Многократное использование контента

◦ Генерация многих видов документов из одной основы Программная документация Эксплуатационная документация Интерактивные системы помощи И т.п.

◦ Поддержка разных форматов (HTML, PDF, ODF, DOCX и т.п.)

} Коммерческие системы ◦ AuthorIT ◦ Adobe Framemaker

} Свободно-распространяемые системы ◦ LaTeX ◦ DocBook ◦ DITA ◦ S1000D

1. LaTeX

} Набор макрорасширений для TeX } Используется система тэгов для форматирования документов } Поддерживается экспорт во все популярные форматы } Последняя версия – LaTeХ 3ɛ

1. DITA

} Darwin Information Typing Architecture } Представлена IBM } OASIS: DITA 1.3, декабрь 2015 г.} XML–документы и DTD-темы } Охватывает весь цикл разработки, выпуска технической документации

1. DocBook

} Стандарт OASIS: DocBook 5.1, ноябрь 2016 г.} Набор XML-тэгов, структурирующих элементы документации } Регламентируется XML-схемой } DocBook предоставляет более 400 тэгов } Независимость содержания от представления } Принцип единого источника } Легко конвертируется в любой формат } Существуют DocBook-редакторы и фильтры для популярных текстовых процессоров

38.Качество процесса проектирования ПО. Модель зрелости процесса проектирования CMM

CMM – Capability Maturity Model (модель полноты потенциала)

} Разработана Software Engineering Institute (SEI) } Версия 1.0 –1991 год } Версия 1.1 -1993 год } Дает обобщенную оценку уровня процесса разработки в организации } Предполагает проведение сертификации на соответствие уровням

Определяет 5 уровней зрелости процесса разработки ПО

1. Начальный

} Процесс не формализован } Нет планирования } Нет контроля } Проект – «черный ящик» } Результат зависит от личных качеств разработчиков } При увольнении сотрудников – проект останавливается или переписывается } По окончании проекта данные о проекте не фиксируются и не оцениваются } Сертификация на 1-ый уровень отсутствует

1. Повторяемый

} Успешный опыт документируется и повторяется } Вводится планирование } Вводится контроль } Вводятся простейшие метрики проектирования } Проекты документируются } Результат по-прежнему зависит от личных качеств разработчиков

1. Определенный

} Все элементы процесса определены и документированы, введены стандарты } Планирование и контроль на основе стандартов компании } Качество ПО не зависит от отдельных разработчиков } По завершении проекта – процесс улучшается, улучшения – документируются } Негласно считается необходимым «минимумом» для компании-разработчика

1. Управляемый

} Вводятся количественные показатели качества и процесса } Более точное планирование проектов. Для определенных областей результат проектирования полностью предсказуем } Имеется накопленная база измерений } Процесс может быть применен к проектам разного типа

1. Оптимизирующий

} Постоянное улучшение и повышение эффективности процессов } Степень улучшения процесса поддается измерению } Эффективность новых технологий может оцениваться количественно

39.Качество процесса проектирования ПО. Модель зрелости процесса проектирования CMMI

Capability Maturity Model Integration (CMMI) — набор методологий совершенствования процессов в организациях разных размеров и видов деятельности } Введена в 2010 году, действующая версия – 1.3

Расширяет CMM набором процессных областей

1. Maturity Level 2 - Managed (управляемый)

◦ Управление требованиями (Requirements Management) ◦ Планирование проекта (Project Planning) ◦ Мониторинг и контроль проекта (Project Monitoring and Control) ◦ Управление договорами с поставщиками (Supplier Agreement Management) ◦ Измерение и анализ (Measurement and Analysis) ◦ Обеспечение качества продукта и процессов (Process and Product Quality Assurance)

1. Maturity Level 3 - Defined (определенный)

◦ Управление конфигурациями (Configuration Management) ◦ Разработка требований (Requirements Development) ◦ Технические решения (Technical Solution) ◦ Интеграция продукта (Product Integration) ◦ Верификация (Verification) ◦ Валидация (Validation) ◦ Фокусирование на процессах организации (Organization Process Focus) ◦ Описание процессов организации (Organization Process Definition) ◦ Организационный тренинг (Organizational Training) ◦ Управление интеграцией проектов (Integrated Project Management) ◦ Управление рисками (Risk Management) ◦ Анализ решений и выбор (Decision Analysis and Resolution)

1. Maturity Level 4 - Quantitatively Managed (количественное управление)

◦ Производительный организационный процесс (Organizational Process Performance) ◦ Количественное управление проектом (Quantitative Project Management)

1. Maturity Level 5 - Optimizing (оптимизирующий)

◦ Организационное управление производительностью (Organizational Performance Management) ◦ Анализ причин и их разрешение (Causal Analysis and Resolution)

