**1. Жизненный цикл ПО. Классификация стратегий конструирования ПО.**

Программа – Данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма.

Программный комплекс–Программа, состоящая из двух или более компонентов и (или) комплексов, выполняющих взаимосвязанные функции, и применяемая самостоятельно или в составе другого комплекса.

Программное обеспечение–Совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ.

Жизненный цикл ПО–Развитие системы, продукта, услуги, проекта или других изготовленных человеком объектов, начиная со стадии разработки концепции и заканчивая прекращением применения.

Программирование–Научная и практическая деятельность по созданию программ.

Основные фазы ЖЦ: анализ и планирование, разработка, документирование, проектирование, тестирование, эксплуатация/сопровождение.

Классификация стратегий конструирования ПО.

1.Однократные

1.1.Определены все требования

1.2.Один цикл конструирования

1.3.Промежуточных версий нет

2.Инкрементные

2.1.Иногда -инкрементно-итеративные

2.2.Определены все требования

2.3.Множество циклов конструирования

2.4.Промежуточные версии могут распространяться

3.Эволюционные

3.1.Иногда -эволюционно-итеративные

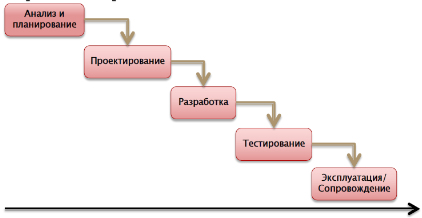
3.2.Определены не все требования

Множество циклов конструирования

Промежуточные версии могут распространяться

**2. Стратегии конструирования ПО(из 1). Классическая модель проектирования ПО**

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.



1.Анализ и планирование

1.1.Сбор требований

1.2.Анализ требований

1.3.Планирование проекта

2.Проектирование

2.1.Разработка архитектуры

2.2.Разработка моделей данных

2.3.Разработка алгоритмов

3.Реализация

3.1.Кодирование

3.2.Отладка

3.3.Тестирование/верификация

4.Сопровождение

4.1.Внедрение

4.2.Эксплуатация

4.3.Внесение изменений

В исходном виде мало подходит к современным проектам. Имеется несколько модификаций.

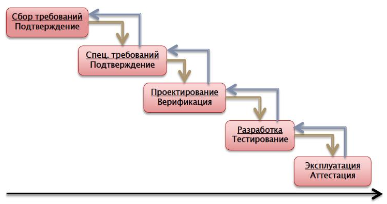
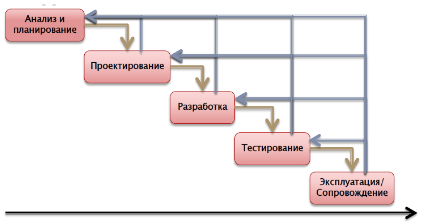
Общепринятая линейная модель

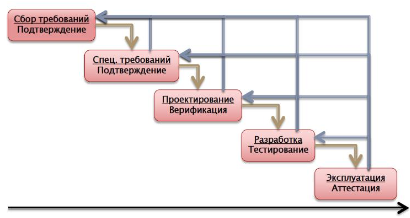
Классическая итерационная

Каскадная модель

Строгая каскадная модель

Строгая каскадная модель





Достоинства:

1.Имеется план и график по всем этапам конструирования

2.Ход конструирования –упорядочен

3.Имеется богатый опыт использования

4.Понятна «большим» заказчикам: государственным, военным, финансовым организациям

Недостатки:

1.Не всегда соответствует реальным проектам(отсутствует гибкость)

2.Часто всех требований на начальном этапе нет

3.Результат доступен только в конце

**3. Стратегии конструирования ПО(из 1). Прототипирование**

Применятся, когда имеются не все требования

1.Позволяет быстро увидеть некоторые свойства продукта

1.1.Удобство

1.2.Внешний вид

1.3.Применимость

2.Часто применятся при проектировании

2.1.Информационных систем

2.2.Программных продуктов с ГПИ(графический пользовательский интерфейс)

3.Используются средства быстрой разработки приложений

Достоинства:

1.Обеспечивает определение полных требований к ПО

2.Наглядно для заказчика

3.Позволяет заказчику рано увидеть основные параметры проекта

Недостатки:

1.По сути не является полным ЖЦ

2.Заказчик может принять макет за продукт

3.Разработчик может принять макет за продукт



**4. Стратегии конструирования ПО(из 1). Инкрементная модель**

Объединяет классический подход и макетирование

1.Весь проект делится на инкременты –версии продукта с определенной функциональностью

2.Для каждого инкремента выполняется:

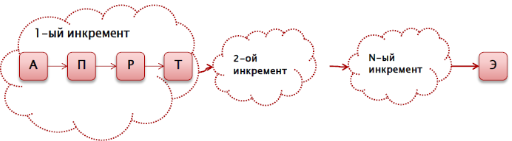
2.1.Анализ

2.2.Проектирование

2.3.Разработка

2.4.Тестирование

3.Результат каждого инкремента – работающий продукт



Достоинства:

1.Имеется план и график по всем этапам конструирования

2.Промежуточные версии доступны заказчику

Недостатки:

1.Часто всех требований на начальном этапе нет

2.Не всегда можно заранее спланировать содержание версий

3.Отсутствует гибкость

**5. Стратегии конструирования ПО(из 1). Спиральная модель**

Предложена Б. Боемом, 1988г и базируется:

1.На классическом ЖЦ

2.На макетировании

Дополнена анализом рисков

Основные компоненты

1.Планирование

2.Анализ

3.Конструирование

4.Реализация

5.Оценивание



Инструментальная спиральная модель

Достоинства:

1.Адекватно отражает эволюционный характер проектирования

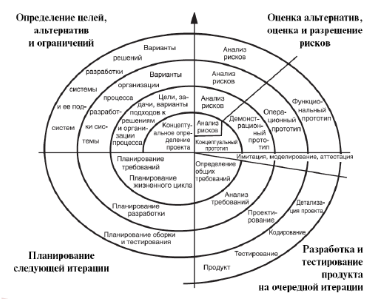
2.Позволяет явно учитывать риски на каждом витке эволюции

3.Использует моделирование и системный подход

Недостатки:

1.Высокие требования к заказчику

2.Трудность контроля времени разработки и управления им



**6. Стратегии конструирования ПО(из 1). Быстрая разработка приложений (RAD)**

RAD = Rapid Application Development

1.Инкрементная стратегия конструирования

2.Использование компонентно-ориентированного конструирования

3.Обеспечение очень короткого цикла разработки (60-90 дней)

4.Ориентирована в основном на разработку информационных систем

Основные этапы:

1.Бизнес-моделирование

Моделируется информационный поток между бизнес -функциями и определяется:

* 1.Какая информация создается
* 2.Кто ее создает
* 3.Кто ее обрабатывает
* 4.Где информация применяется

2.Моделирование данных

По информационному потоку формируется набор объектов данных

* 1.Определяются свойства объектов
* 2.Специфицируются отношения между объектами

3.Моделирование обработки

Определение преобразований объектов данных. Создаются описания для

* 1.добавления объектов данных
* 2.модификации объектов данных
* 3.удаления объектов данных
* 4.поиска объектов данных

4.Генерация приложения

Использование ЯП 4-го поколения

* 1.Использование готовых компонентов
* 2.Создание повторно используемых компонентов
* 3.Использования средства aвтоматизации

5.Тестирование и объединение

Тестирование упрощается из-за повторного использования компонентов

* 1.Они не требуют автономного тестирования
* 2.Используется интеграционное тестирование

Ограничения:

* Область применения –проектирование информационных систем
* 2.Производительность не является критичной
* 3.Неприменимо для задач реального времени
* 4.Можно привлечь достаточно разработчиков
* 5.Отсутствуют технические риски

**7. Стратегии конструирования ПО(из 1). Rational Unified Process**

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

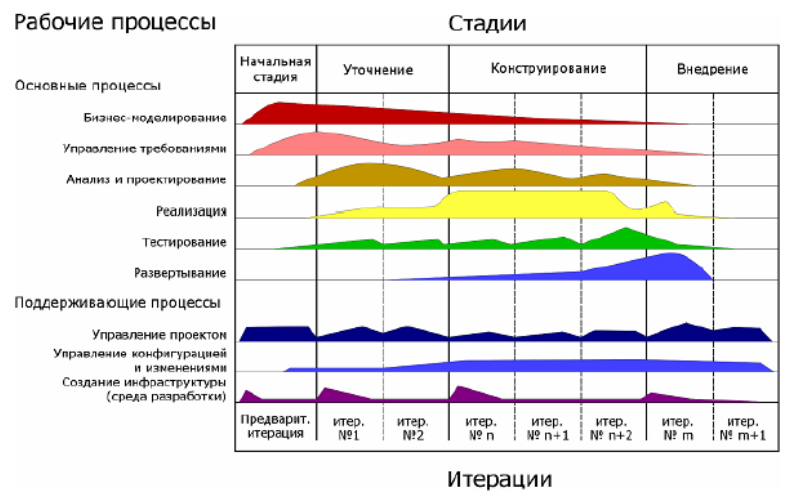
Базируется на широком использовании UML

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

1.Бизнес-моделирование

2.Управление требованиями

3.Анализ и проектирование

a. Создание статического и динамического представления системы

4.Реализация

a. Создание программного кода

5.Тестирование

a. Проверка системы в целом

Выводы

Наиболее продуманная методология

Подходит для больших и очень больших проектов (реже средних)

Требует высокой квалификации участников

**8. Стратегии конструирования ПО (из 1). 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

Динамика определяется:

* Непрерывностью связи с заказчиком
* Простотой –выбирается простейшее решение
* Быстрой обратной связью
* Профилактикой проблем

Методы XP

1.Planning game (Игра в планирование)

Заказчик:

Объем работ

Приоритет

Композиция версий

Сроки выпуска версий

Разработчик:

Временные оценки

Последствия

Процесс

Подробный график работ

2.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

3.Metaphor (Метафора)

Глобальное «видение» проекта, понятное всем «Замена» большой архитектуры

В данном контексте – основная идея проекта, сведения об архитектуре

4.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

5.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)

6.Refactoring (Рефакторинг)

Изменение программы для упрощения добавления новой возможности

Изменение программы после добавления новой функциональности

Программы до и после рефакторинга функционально эквивалентны

7.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

8.Collective ownership (Коллективное владение)

Код –общая собственность

При необходимости код модифицируется немедленно, независимо от авторства кода

9.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

10.40-hour week (40-часовая неделя)

Сверхурочные –крайне нежелательны

Если постоянно требуется переработка –неправильно организован проект

Отпуск –обязателен

11.On-site customer (Локальный заказчик)

В составе команды –представитель заказчика

Представитель –пользователь системы

Представитель отвечает на вопросы разработчиков и расставляет мелкие приоритеты

12.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

Список работ, которые необходимы выполнить

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
* 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 до9) 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

Бережливая разработка приложений

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 бережливой разработки

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

**10. Управление требованиями. Требования в программных проектах. Проблемы определения требований.**

Требование по IEEE 1990 (IEEE standard glossary of software engineering terminology):

1.Условие или возможность, необходимые пользователю для решения его задач или достижения цели.

2.Условие или возможность, которым должна отвечать или которыми должна обладать система или ее компонента, чтобы удовлетворить контракт, стандарт, спецификацию или иной формальный документ.

3.Документированное представление условия или возможности, указанное в (1) или (2)

Свойства требования

* Корректность (correct)
* Однозначность (unambiguous)
* Полнота(complete)
* Непротиворечивость (consistent)
* Приоритезация (prioritized)
* Проверяемость (verifiable)
* Модифицируемость (modifiable)
* Отслеживаемость (traceable)

Виды требований

Функциональные требования

Бизнес-требования

Пользовательские требования

Нефункциональные требования

Ограничения

Требования к качеству

Управление требованиями

Управление изменениями

* Предложение изменений
* Анализ изменений
* Принятие решений
* Обновление требований
* Обновление планов

Контроль версий

* Определение схемы идентификации версий
* Определение версий спецификаций требований
* Определение версий отдельных требований
* Контроль состояния требований
* Определение состояния требований
* Регистрация состояния требования

Прослеживание требований

* Определение связей с другими требованиями
* Определение связей с другими элементам системы

Управление версиями 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

Управление сосотояниями требований

|  |  |
| --- | --- |
| Состояние | Определение |
| Предложено | Требование было выставлено авторизованным источником |
| Одобрено | Требование было проанализировано и одобрено для определенной версии. |
| Реализовано | Код, реализующий требование, был написан |
| Проверено | Корректная функциональность донного требования была подтверждена версией продукта. Требование может быть прослежено до варианта тестирования. |
| Удалено | Ранее одобренное требование было исключено из базисного списка. Причина удаления –задокументирована |
| Отклонено | Предложенное требование –отклонено. Причина отклонения –задокументирована |

**11. Разработка требований. Выявление и анализ требований.**

Разработка требований

Выявление требований

Анализ требований

Спецификаци требований

Выявление требований

Заинтересованные лица

* Заказчики≠пользователь
* Менеджеры
* Пользователи
* Операторы
* Менеджеры
* Разработчики
* Служба поддержки
* Другие лица

Планирование

* Цели выявления требований
* Стратегии и процессы выявления требований
* Результаты усилий по выявлению требований
* Оценки календарного плана и ресурсов
* Риски, связанные с выявлением требований

Проблемы определения требований:

* Ожидания пользователей
* Умение оценить противоречивые требования
* Недостаточные требования
* Умение понять требования пользователей

Способы выявления требований

* Семинары
* Исследования
* Интервью
* Создание прототипов
* Фокус-группы

Проблемы:

* Сложности формулирования требований
* Терминология
* Используемые неявные допущения
* Предвзятые решения

Анализ требований

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

Анализ требований –сходящийся процесс:

* Уточняет данные
* Структурирует информацию
* Устанавливает приоритеты

Результат анализа –спецификация требований

Уточнение требований

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

Приоритеты

Должны участвовать все заинтересованные лица проекта:

* Заказчики
* Пользователи
* Разработчики

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:

|  |  |  |
| --- | --- | --- |
|  | Важно | Не важно |
| Срочно | Высший приоритет | Низший приоритет |
| Не срочно | Средний приоритет | Не стоит делать |

**12. 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 требований

Группирование требований

Требования объединяются в родственные группы

Иерархическая структуризация требований

Подчинение

Уточнение

**13. Документирование требований. Виды документов.**

Способы документирования требований

* Документы на естественном языке
* Графические модели
* Диаграммы
* Графы
* Схемы
* Потоки
* Формальные спецификации

Типы документов

Создаются все или некоторые из документов:

Состав и распределение работ

Спецификация требований

Концепция эксплуатации

Начальный план разработки ПО

Критерии принятия работ

**14. Изменения требований.**

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 неправильно поняты
* Требования плохо определены
* Требования не были поняты
* Сработали архитектурные риски

Условия возможности изменений

Водопадные стратегии –практически невозможно

Инкрементные стратегии –возможно с некоторыми ограничениями

Эволюционные стратегии –возможно

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 лица 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

**15. Планирование и управление требованиями. Прослеживание требований.**

Цели:

Контроль версий требований

Контроль состояния требования

Прослеживаемость требований

Управление изменениями требований

Совершенствование процессов управления

Управление требованиями

Управление изменениями

* Предложение изменений
* Анализ изменений
* Принятие решений
* Обновление требований
* Обновление планов

Контроль версий

* Определение схемы идентификации версий
* Определение версий спецификаций требований
* Определение версий отдельных требований
* Контроль состояния требований
* Определение состояния требований
* Регистрация состояния требования

Прослеживание требований

* Определение связей с другими требованиями
* Определение связей с другими элементам системы

Управление версиями 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

Управление сосотояниями требований

|  |  |
| --- | --- |
| Состояние | Определение |
| Предложено | Требование было выставлено авторизованным источником |
| Одобрено | Требование было проанализировано и одобрено для определенной версии. |
| Реализовано | Код, реализующий требование, был написан |
| Проверено | Корректная функциональность донного требования была подтверждена версией продукта. Требование может быть прослежено до варианта тестирования. |
| Удалено | Ранее одобренное требование было исключено из базисного списка. Причина удаления –задокументирована |
| Отклонено | Предложенное требование –отклонено. Причина отклонения –задокументирована |

**16. Ресурсы в программных проектах. Управление ресурсами**

Управление ресурсами

Ресурс -объект проекта, подлежащий управлению и планированию

Виды ресурсов: Сотрудники, Рабочее время, Оборудование, Машинное время, Программное обеспечение

Управление прочими ресурсами

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

**17. Роли участников в программных проектах**

Сотрудники и роли

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.

Роли в процессе разработки программных проектов

**Основные:** 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

Заказчик

Инициирует разработку, участвует в сборе требований, участвует в разработке спецификации требований, принимает результаты разработки

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

Менеджер проекта

Внешние функции:

взаимодействие с инициаторам и проекта: заказчиком и Планировщиком ресурсов

Внутренние функции:

распределяет задачи среди членов команды, организует выполнение проекта

Архитектор

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

Руководитель команды

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

Разработчик

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

Тестер

Тестировщик, Quality Assurance (QA - Гарантия качества), 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

Разработчик документации

Технический писатель, technical writer, Разработка программной

Документации, Разработка эксплуатационной документации, Ведение информационной поддержки процесса разработки

Пользователь

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

Эксперт предметной области

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

Специалист по пользовательскому интерфейсу и эргономике

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

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

Библиотекарь

Ведет библиотеку проекта, Контролирует соответствие выпускаемого продукта принятым

стандартам

**18. Проектные активности программных проектов**

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

Проект –самостоятельно управляемый элемент разработки

Нормальный результат программного проекта – программный продукт

Проектные активности: Выполнение задач из ТЗ (-подчиненных проектов, работ), Реализация изменений, Исправление дефектов

**19. Временные сущности программных проектов**

Временные сущности: Этапы (stage), Вехи (milestone)

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

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.

**20. Визуализация плана. Наблюдение за программным проектом**

Визуализация –основной способ планирования, контроля и наблюдения.

Существует два основных подхода: Диаграммы Ганта, Диаграммы PERT

Гранта: Генри Гант, 1910 г. Использовалась при управлении во время проектирования кораблей

Программные продукты: GanttProject, OpenProj, MS Visio, MS Excel, MS Project, Trac

Диаграммы PERT: Program Evaluation and Review Technique, 1958г.

Другие названия –сетевой график, сетевой план-график.

Наблюдение за проектом

Активности: Задачи, дефекты, фиксации изменений, Ресурсы: Сотрудники, ...

Временные сущности: Этапы, вехи, критический путь

Виды срезов: Задачи, Сотрудники, Вехи, Дефекты, Фиксации изменений (коммиты) в СКВ, Критический путь

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

Задачи: Срез по сотрудникам, Текущие задачи сотрудника, Отставание от графика сотрудника, Общее количество дефектов, относящихся к сотруднику, Количество незакрытых дефектов, относящихся к сотруднику.

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, Срез по фиксациям изменений(коммитам) В СКВ, Среднее число фиксаций на сотрудника за единицу времени, Равномерность фиксаций у сотрудников

Срез по критическому пути: Сотрудники в критическом пути, Задачи в критическом пути, Временные запасы в критическом пути

**21. Понятие дефекта программного обеспечения. Характеристики дефектов**

Дефект –обнаруженная в процессе разработки, тестирования или эксплуатации ошибка в разрабатываемом приложении. Ошибка может заключаться: В явном нарушении спецификации, В неявном нарушении спецификации, Исправление обнаруженного дефекта – самостоятельная проектная активность.

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

Контекст дефекта: 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

Содержание дефекта: 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.

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

Категория дефекта: Описывает тип найденного дефекта. Возможные категории: Функциональный дефект, Дефект документации, Дефект требований, Предложение по усовершенствованию \*, Новая функция \*

Серьезность дефекта: 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.

Автор дефекта: 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

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.

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.

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

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

Временные параметры устранения дефекта: Желаемое время, когда требуется устранить дефект. Желаемое версия проекта, к которой требуется устранить дефект

Резолюция на дефект: Необязательная текстовая реакция ответственного за исправление

Может сопровождать переход дефекта из одного состояния в другое

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

**22. Жизненный цикл дефекта.**

Изменение характеристик дефекта

◦Переход из одного состояния в другое состояние

◦Изменение ответственного за исправление

◦Изменение ответственного за проверку

◦РЕДКО: изменение автора

◦Изменение контекста

◦РЕДКО: Изменение серьезности

◦Изменение срочности

◦РЕДКО: Изменение категории

◦Изменение содержания

◦Изменение резолюции

◦Изменение способа обхода

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

**23. Системы управления дефектами.**

Другие названия: Управление изменениями и дефектами, Трассировка изменений, Трассировка ошибок, Отслеживание ошибок, Управление инцидентами, Управление рекламациями ◦и т.п.. Bug tracking, issue tracking

Коммерческие: Borland StarTeam, Atlassian JIRA, JetBrains YouTrack

Свободно распространяемые: Mozilla Bugzilla, Trac, Redmine, Jira, Redmine

Интегрированные в серверы хостинга проектов (BitBucket, GitHub и т.п.)

**24. Управление риском в программных проектах: идентификация, анализ, ранжирование**

Начало: 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.

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)

Заинтересованные лица: Заказчики≠пользователь, Менеджеры, Пользователи, Операторы

Менеджеры, Разработчики, Служба поддержки, Другие лица

Планирование: Цели выявления требований, Стратегии и процессы выявления требований, Результаты усилий по выявлению требований, Оценки календарного плана и ресурсов Риски, связанные с выявлением требований

Проблемы определения требований: Ожидания пользователей, Умение оценить противоречивые требования, Недостаточные требования, Умение понять требования пользователей

Способы выявления требований: Семинары, Исследования, Интервью, Создание прототипов, Фокус-группы

Проблемы: Сложности формулирования требований, Терминология, Используемые неявные допущения, Предвзятые решения.

**25. Управление риском в программных проектах: планирование, разрешение, наблюдение.**

Планирование: Спринт (Sprint) 30-тидневный (обычно) промежуток за который выполняется реализация заданной функциональности. Планирование спринта: Происходит в начале спринта, Ежедневная встреча разработчиков Демонстрация: Происходит в конце спринта.

Наблюдение за проектом: Активности: Задачи, дефекты, фиксации изменений. Ресурсы:

Сотрудники, …Временные сущности: Этапы, вехи, критический путь. Виды срезов: Задачи, Сотрудники, Вехи, Дефекты, Фиксации изменений (коммиты) в СКВ, Критический путь

Срез по задачам: Сотрудники, занятые решением задачи, Соответствие задач – графикам

Процент завершенности по задачам проекта, Общее количество дефектов задачи, Количество незакрытых дефектов

Задачи: Срез по сотрудникам, Текущие задачи сотрудника, Отставание от графика, сотрудника, Общее количество дефектов, относящихся к сотруднику, Количество незакрытых дефектов, относящихся к сотруднику.

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, Срез по фиксациям изменений (коммитам) В СКВ, Среднее число фиксаций на сотрудника за единицу времени, Равномерность фиксаций у сотрудников

Срез по критическому пути: Сотрудники в критическом пути, Задачи в критическом пути, Временные запасы в критическом пути

Анализ причин и их разрешение (Causal Analysis and Resolution).

**26. Предпосылки для версионирования ПО. Ветвление.**

Системы управления версиями

(VCS -Version Control System), Системы контроля ревизий (RCS - Revision ControlSystem), Системы управления исходным кодом (SCM –Source Code Management)

СКВ предназначены для автоматизации групповой работы и управления версионированием проектов. СКВ обеспечивают: Репозиторий (или репозитории) хранения проектов, Стандартные операции обеспечения групповой работы, Клиенты для выполнения операций

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.

Причины разветвления версий файла: Развитие нескольких версий проекта Поставленных заказчику Разрабатываемых, Наличие нескольких конфигураций проекта Для разной аппаратуры, Для разных операционных систем Разработка новой (экспериментальной) функциональности.

**27. Системы контроля версий. Типы СКВ. Общие принципы организации.**

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, Сравнение версий, Объединение разрозненного кода

Системы управления версиями

(VCS -Version Control System), Системы контроля ревизий (RCS - Revision ControlSystem), Системы управления исходным кодом (SCM –Source Code Management)

СКВ предназначены для автоматизации групповой работы и управления версионированием проектов. СКВ обеспечивают: Репозиторий (или репозитории) хранения проектов, Стандартные операции обеспечения групповой работы, Клиенты для выполнения операций

Типы СКВ: Централизованные СКВ, Единое централизованное хранилище, Клиент-серверный доступ. Примеры: CVS, Subversion, MS Visual Source Safe и т.п.

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.

**28. Системы контроля версий. Типовые операции**

См. вопрос 27

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 по 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

**29. Сборка программных проектов. Проблемы при сборке программных проектов**

См. вопрос 30

Проблемы с исходными текстами, Отсутствие всех исходных текстов, Некорректное расположение исходных текстов, Неправильные версии исходных файлов.

Проблемы с подключаемыми файлами: 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.

Проблемы с используемыми библиотеками: 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.

Проблемы с процедурами сборки: Отсутствие процедуры сборки проекта, Отсутствие процедур сборки компонентов проекта, Несоответствие версии процедуры сборки.

Проблемы со средствами (утилитами) 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.

Проблемы с системной средой и аппаратной платформой сборки: 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.

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

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), Исходными для сборки являются: Исходные файлы ЯП, Шаблоны, Конфигурационные файлы, Ресурсные файлы, Файлы документации, И т.п.

Окружение для сборки: 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 И т.п.

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 make tools, make, nmake, Apache Ant, Apache Maven

Сборка компонентов проекта: Для каждого компонента проекта должна быть сформирована локальная процедура сборки. Например, makefile. В каждом компоненте проекта в 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.

Утилиты сборки Make tools:, GNU Make, Nmake, Ant, Maven, Gradle, Генераторы make -файлов, GNU Automake, Makedep, Makedepend.

Средства управления сборкой: FinalBuilder, TeamCity, Visual Build,Codefast PerfectBuild

… (около 100)

**31. Непрерывная интеграция.**

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 тэгом 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 : По расписанию Обычно запуск производится ночью (nigthly build, daily build), Проводится запуск полного набора тестов

По каждому обновлению файлов репозитория: 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

По запросу.

Промышленные средства непрерывной интеграции: CruiseControl, CruiseControl.NET, Hudson/Jenkins, BuildBot, Bitten —плагин для Trac, MS Team Foundation Server

**32. Качество программного обеспечения. Характеристики качества.**

Характеристики качества – набор свойств программной продукции, по которым её качество описывается и оценивается: Функциональность (Functionality), Надежность (Reliability), Практичность (Usability), Эффективность (Efficiences), Сопровождаемость (Maintainability), Мобильность (Portability)

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

Характеристики качества: Функциональная пригодность (functional suitability), Уровень производительности (performance efficiency), Совместимость (compatibility), Удобство использования (usability), Надежность (reliability), Защищенность (security), Сопровождаемость (maintainability), Переносимость (portability)

Функциональная пригодность: Степень, в которой продукт или система обеспечивают выполнение функции в соответствии с заявленными и подразумеваемыми потребностями при использовании в указанных условиях. Подхарактеристики: функциональная полнота (functional completeness): степень покрытия совокупностью функций всех определенных задач и целей пользователя, функциональная корректность (functional correctness): степень обеспечения

продуктом или системой необходимой степени точности корректных результатов. функциональная целесообразность (functional appropriateness): степень функционального упрощения выполнения определенных задач и достижения целей.

Уровень производительности: Производительность относительно суммы использованных при определенных условиях ресурсов.

Подхарактеристики: Временные характеристики (time behaviour): степень соответствия требованиям по времени отклика, времени обработки и показателей пропускной способности продукта или системы. Использование ресурсов (resource utilization): степень удовлетворения требований по потреблению объемов и видов ресурсов продуктом или системой при выполнении их функций. Потенциальные возможности (capacity): степень соответствия требованиям предельных значений параметров продукта или системы.

Совместимость: Способность обмениваться информацией с другими продуктами, системами или компонентами, и/или выполнять требуемые функции при совместном использовании одних и тех же аппаратных средств или программной среды.

Подхарактеристики: Сосуществование (совместимость) (co-existence): способность продукта совместно функционировать с другими независимыми продуктами в общей среде с разделением общих ресурсов и без отрицательного влияния на любой другой продукт. Функциональная совместимость (интероперабельность) (interoperability): способность двух или более систем, продуктов или компонент обмениваться информацией и использовать такую информацию.

Удобство использования: Степень, в которой продукт или система могут быть использованы определенными пользователями для достижения конкретных целей с эффективностью, результативностью и удовлетворенностью в заданном контексте использования.

Подхарактеристики: определимость пригодности (appropriateness recognizability): возможность пользователейпонять, подходит ли продукт или система для их потребностей, сравним ли с функциональной целесообразностью изучаемость (learnability): возможность использования продукта или системы определенными пользователями для достижения конкретных целей обучения для эксплуатации продукта или системы с эффективностью, результативностью, свободой от риска и в соответствии с требованиями в указанном контексте использования. управляемость (operability): наличие в продукте или системе атрибутов, обеспечивающих простое управление и контроль. защищенность от ошибки пользователя (user error protection): уровень системной защиты пользователей от ошибок. эстетика пользовательского интерфейса (user interface aesthetics): Степень "приятности" и "удовлетворенности" пользователя интерфейсом взаимодействия с пользователем. доступность (accessibility): возможность использования продукта или системы для достижения определенной цели в указанном контексте использования широким кругом людей с самыми разными возможностями.

Надежность: Степень выполнения системой, продуктом или компонентом определенных функций при указанных условиях в течение установленного периода времени.

Подхарактеристики: завершенность (maturity): степень соответствия системы, продукта или компонента при нормальной работе требованиям надежности. готовность (availability): степень работоспособности и доступности системы, продукта или компонента отказоустойчивость (fault tolerance): способность системы, продукта или

компонента работать как предназначено, несмотря на наличие дефектов программного обеспечения или аппаратных средств. восстанавливаемость (recoverability): способность продукта или системы восстановить данные и требуемое состояние системы в случае прерывания или сбоя.

Защищенность: Степень защищенности информации и данных, обеспечиваемая продуктом или системой путем ограничения доступа людей, других продуктов или систем к данным в соответствии с типами и уровнями авторизации.

Подхарактеристики: Конфиденциальность (confidentiality): обеспечение продуктом или системой ограничения доступа к данным только для тех, кому доступ разрешен.

Целостность (integrity): степень предотвращения системой, продуктом или компонентом

несанкционированного доступа или модификации компьютерных программ или данных.

Неподдельность (non-repudiation): степень, с которой может быть доказан факт события или действия таким образом, что этот факт не может быть отвергнут когда-либо позже.

Отслеживаемость (accountability): степень, до которой действия объекта могут быть прослежены однозначно. Подлинность (authenticity): степень достоверности тождественности объекта или ресурса требуемому объекту или ресурсу.

Сопровождаемость: Результативность и эффективность, с которыми продукт или система могут быть модифицированы предполагаемыми специалистами по обслуживанию.

Подхарактеристики: Модульность (modularity): степень представления системы или компьютерной программы в виде отдельных блоков таким образом, чтобы изменение одного компонента оказывало минимальное воздействие на другие компоненты. Возможность многократного использования (reusability): степень, в которой актив может быть использован в нескольких системах или в создании других активов. Анализируемость (analysability): степень простоты оценки влияния изменений одной или более частей на продукт или систему или простоты диагностики продукта для выявления недостатков и причин отказов, или простоты идентификации частей, подлежащих изменению. Модифицируемость (modifiabiIity): степень простоты эффективного и рационального изменения продукта или системы без добавления дефектов и снижения качества продукта. Тестируемость (testability): степень простоты эффективного и рационального определения для системы, продукта или компонента критериев тестирования, а также простоты выполнения тестирования с целью определения соответствия этим критериям.

Переносимость: Степень простоты эффективного и рационального переноса системы, продукта или компонента из одной среды (аппаратных средств, программного обеспечения, операционных условий или условий использования) в другую.

Подхарактеристики: Адаптируемость (adaptability): степень простоты эффективной и рациональной адаптации для отличающихся или усовершенствованных аппаратных средств, программного обеспечения, других операционных сред или условий использования. Устанавливаемость (installability): степень простоты эффективной и рациональной, успешной установки и/или удаления продукта или системы в заданной среде. Взаимозаменяемость (replaceability): способность продукта заменить другой

конкретный программный продукт для достижения тех же целей в тех же условиях.

**33. Методы обеспечения качества ПО. Формальная верификация.**

Верификация - подтверждение на основе представления объективных

свидетельств того, что установленные требования были выполнены

Валидация - подтверждение на основе представления объективных

свидетельств того, что требования, предназначенные для конкретного использования или применения, выполнены, декларируемые свойства и характеристики подтверждаются, а поставленная цель (предназначение системы, комплекса, устройства и т. д.) достигнута.

Методы обеспечение качества программных систем: По используемым формализмам: формальные методы, Неформальные методы, По необходимости запуска анализируемой программы: Динамические, Статические, Гибридные. По уровню автоматизации: Ручные, Автоматизированные, Автоматические.

Динамические методы: Тестирование, Профилирование, Динамический анализ: Мониторинг, Анализ трасс исполнения, Контрактное программирование, Статические методы: Формальная верификация: Дедуктивная верификация, Model checking (методы проверки модели), Статический анализ, Трансформации программ, Рефакторинги, Модификации, Аудит

Верификация - подтверждение соответствия конечного продукта функциональной спецификации

Формальная верификация – доказательство корректности с

помощью формальных методов

Используемые методы и мат. аппарат: Пропозициональные логики, Темпоральные логики, Формальные семантики, Формальные преобразования программ, Формальные спецификации, Логика Хоара, Сепарационная логика (separation logic) И т.п.

Наиболее известные подходы: Верификация методом Хоара (на основе троек Хоара), Верификация по Флойду

Достоинства: В случае успеха – в программе нет ошибок!

Недостатки: Формальные спецификации на порядок сложнее программ

Для большинства программ задача формального доказательства корректности – очень трудоёмка. Для некоторых случаев – задача формального доказательства корректности – неразрешима. В реальных системах при формальной верификации рассматривают часть системы и частичные спецификации. Редко применяется для обеспечения качества программных систем общего назначения

**34. Методы обеспечения качества ПО. Метод проверки моделей.**

Проверка модели, проверка на модели, model checking. Метод формальной верификации для систем с конечным числом состояний. Позволяет проверить, удовлетворяет или нет система некоторому свойству (требованию).

Верификация по методу Model Checking. Исследуемая система приводится к модели с конечным числом состояний (например, модель Крипке). Проверяемые свойства представляются формулами темпоральной логики (LTL, ALTL, CTL, CTL\* и т.д.). Проверка модели – формальная проверка выполнимости формулы на модели. Результат проверки: Формула выполняется, Формула не выполняется. Контрпример. Существуют методы проверки систем с 10^(100-200) состояний

Ограничения: Проверяются свойства, связанные только с корректностью смены состояний. Не все свойства представляются в виде темпоральных формул. В общем случае задача - NP-полная. В общем случае неформализуется переход от реальной системы к модели с конечным числом состояний.

Обнаружение программных дефектов: Динамические методы: Тестирование, Динамический анализ. Статические методы: Статический анализ, Верификация (частично)

**35. Методы обеспечения качества ПО. Статический анализ ПО.**

Использует исходный код ПО для анализа. Применяется для: Форматирования программ, Вычисления программных метрик, Оптимизации программ, Распараллеливания программ, Преобразования программ, Обфускации программ, Деобфускации программ, Обнаружения дефектов

Цель – обнаружение дефектов в программном коде Использует исходный код ПО для анализа. Позволяет проанализировать все возможные трассы исполнения. Позволяет проанализировать все наборы входных данных. Может быть полностью автоматизирован. Позволяет обнаружить нефункциональные дефекты

Статистический анализ: Используемые методы: Интервальный анализ, Поиск достижимости, Анализ указателей, Ресурсный анализ, Сигнатурный анализ

Достоинства СА: Обнаружение дефектов на ранних стадиях, Сокращение стоимости разработки, отладки, тестирования, сопровождения.

Недостатки СА: невозможность обнаруживать функциональные ошибки, Недостаточность информации о путях выполнения -> наличие ложных обнаружений, Невозможность обнаружить все ошибки статически, Высокие требования к вычислительным ресурсам