**Программная инженерия (прикладная наука не computer science)** – инженерная дисциплина, изучающая методы создания, развития и анализа свойств программного обеспечения.

Объект изучения – программная система (информационный а не математический объект)

Программная система отличается от программы большим количеством людей:

1)решается группа задач а не конкретная

2)программные системы живут долго и потому есть необходимость ее развития

3)экономические вопросы появляются – ограниченность бюджета и времени.

**Способы борьбы со сложностью программирования системы**

1)Модульность –

а)разбить задачи на части

б)решение подзадач

в)сборка обратно

г)выделение интерфейса

д)сокрытие внутренней реализации

**1**

2)абстракции (нужно выделить важное и не важное)

состав важного и неважного может меняться на ходу

3)многократное использование

**Пример** системаISO/OSI – стек протоколов

1)физический

2)канальный

3)сетевой

4)транспортный

5)прикладной

Чтобы обеспечить модульность и при этом не усложнить всё, разбивать на модули надо с умом и в этих модулях должны быть “интерфейсы” для его пользования.

Интерфейсы – фиксируемы и не могут меняться.

Интерфейсы должны быть:

1)адекватность – т.е. должны содержать все необходимое в зависимости от задачи, решаемой модулем.

2)полны – все необходимые вещи есть в интерфейсе

3)минимальны – нет ничего лишнего

4)простота и понятность

выгода ~ качество

ресурсы время люди деньги

Качество => удобство развития

Любая система должна демонстрировать хоть какую-нибудь функциональность даже при некоректной работе. (например не сотрет БД) **2**

Технологический процесские процессы разбаротки ПО

* виды деятельности
* артефакты(документы модели код)  **СОСТАВЛЯЮЩИЕ ПРОЦЕССОВ**
* роли

**Пример**:

1)пришел заказчик => нужно договориться =>

Заказчик и Руководитель =>роли

“Инициализация проекта” => деятельность

“Концепция” => артефакты

“Оценка проекта” => артефакты

2)”Анализ и фиксация деятельности”

(6) <= “Тех задание” или “Спецификация требований”

заказчик/эксперты / аналитик

(более общее)

3)”Архитектурное проектирование ” => Архитектура <= правила, как компоненты

Структура <=выделяются и взаимодействуют+

архитектор/руководитель проекта +контр набор компонентов.

детальные планы

(для контроля планирования проекта)

4)”Детальное проектирование” =>”Детальный проект”

“Проекторование пользовательсткого интерфейса” =>”Структура UI”

архитектор/разработчики UI

5)Построение ПО

Документация

6)Тестирование => “Отчет”

тестировщики

7)”Передача в эксплуатацию”

старший разработчик

анатилик

заказчик

**3**

Жизненный Цикл ПО

Идея создания Эксплуатация Эксплуатация Отказ от эксплуатации

| | | | | | | | | | |

доработка доработка

20-25% всех потраченных ресурсов

**Модели жизненного цикла**

1)каскадная (водопадная)

когда деятельность идет одна за другой в технологическом процессе на предыд стр

только в очень простых проектах

2)итеративная

самая практичная

Инкрементальная – отдельно идет разработка для разных модулей

3)Спиральная

Анализ и оценка решений

Выработка решений

Планировщие

Реализация

**Пример**:

RUP – Rational Unified Process

Системы

тестирования

V – модель - Требования

Детали дизайна

Архитектура

**4**

Тот проверяется

соответствие

UNIT тестирование

Интерфейс тестирования

**Стандарты**

IEEE 1074 – американский институт электроники и эелктротехники

ISO – International Standartization Organization

12207 -

15288 – процесс разработки программно-аппаратной системы

15504(SPICE) – регулирует оценку процесса разработки

указ виды деятельности

CMM/CMMI – дает +5 уровней зрелости и организации

(чем больше уровней зрелости тем больше состояний в процессе разработки)

1)нет требований

2)повторяемый(управляемый)

(если ведется какая-то оценка ресурсов)

3)определенный

(четко прописан процесс разработки)

4)управляемый (управляемый метриками)

ним. Оценки по затратам ресурсов

ним. Оценки качества

делаются прогнозы основываюсь на

5)совершенствующийся

есть инновации в процессе разработки

**Пример**: ISO 12207

Процессы:

Управление договорами

Тут все работы по приобрет. использ. сторонных ПО

Процессы обеспечения

Обеспечение возможности работы разных проектов например –

1)управл. инфраструктуры

2)управл. человеч. Ресурсов

3)управл. Качеством – проверка, что есть контроль качества

Проектная деятельность

1)планирование

2)оценка и текущий контроль

3)управление рисками

4)принятие решений

5)управлений конфигурациями

6)измерение затрат

**5**

Процессы построения ПО

1)анализ требований

2)архитектурное проектирование

3)детальное проектирование

4)построение ПО/реализация

5)интеграция

6)квалификационное тестирование

Поддерживающие

1)управление документами

2)управление конфигурациями

3)обеспечение качества

4)верификация и валидация

есть четкие нет четких

критерии проверки критериев

5)экспертизы и аудиты

6)разрешение проблем

Построение системы

Многократное использование

Как использовать несколько раз докум./модули и прочее

RUP/XP

Rational Unified Process / Extreme programming

Жесткий процесс agile – гибкий процесс разработки

Разработки

Пытается оторвать привязывается к имеющимся людям по максимальному

Разработку от самого использованию их потенциал., но

Продукта, чтобы не было привязки после работы тут остается мало докум.

К программистам

И те и другие нацелены на постоянные изменения, в процессе разработки, и после.

Поэтому есть:

1)итерации

2)использование use case (это для RUP)

**6**

т.е. все требования должны быть описаны в виде use case

3)архитектура – ее использование для планирования всех действий

4)модели – они описывают все принятые решение, чтобы мог вникнуть посторонний человек

1)начало проекта – нужно определять с бюджета, ресурса, задач, использованных технологий, команды ~10%(трудовые) ~5%(временные)

2)проектирование – выработка архитектуры ~20%(трудовые) ~30%(временные)

3)построение (разработка(разработка кода и тесты)) ( construction)

~65%(трудовые) ~5(50)%(временные)

4)внедрение(передача заказчику – документация/обучение заказчика/…)

~5%(трудовые) ~15%(временные)

**4 фазы каждой итерации**

итераций может быть много

Модели:

1)варианты использования use case

обычно рисуются овалами, сценарии описывают отдельно роли

(описывают требования к системе в виду сценариев работы)

у основных сценариев могут быть ответвления

2)модель анализа(концептуальная)

набор классов/сущностей – работающие в системе

3)проектирование

1. интерфейсы

2. управляющие ОСНОВНЫЕ КЛАССЫ

3.классы данных

4)реализации

набор компонент, в которых классы уже собраны,

компоненты друг от друга достаточно независимы

описание того, как система расположена на железе –

т.е. это набор узлов(?) и периферических устройств.

5)развертывание

описание того, как система расположена на железе –

т.е. это набор узлов(?) и периферических устройств.

6)тестирование.

1. Состоит из тестовые вариантов – имеют более или менее

абстрактное описание.

Для каждого сценария(каждой альтернативы) строится тестовый вариант

Альтернатива в тестах может быть только, если система недетерминирована.

2. Тестовые процедуры –

Тут все четко определено(не абстрактное описание)

Виды деятельности (в RUP - дисциплины)

1)Бизнес моделирование (моделирование предметной области)

строится модель анализа

**7**

2)определение требований – описание вариантов использования и их реализация в моделях

анализа

3)анализ и проектирование

создается модель проектирования

4)реализация системы – разработка кода

5)тестирование – разработка тестов и тестирование

6)развертка – передача в руки пользователю

Поддерживающее деятельность

7)1)управление конфигурацией

определение разных версий/составляющих модулей

8)2)управление проектом

все административные работы

9)3)управление средой проекта

подготовка рабочего места/средств разработки/…

1

2

3

4

5

6

4

3

2

1

4

3

2

1

Пункты с конца предыдущего листа

XP

В XP есть 4 принципа(они отражают agile методы)

1)люди важнее процессов

(то есть отдается приоритет людям, а не инструментам)

2)система важнее документация

3)сотрудничества важнее контракт

с заказчиком

( то есть привязыватьмя надо к тому, что заказчик хочет, а не то что описано в

тех зад.)

4)изменение важнее планов

**Структура итераций**

**8**

Основная задача для системы – определить архитектурный стиль системы

Сценарий исп. для

разработка

тесты

версия

Метафора

системы

Планирование

(фиксирование основного решения и

кто-что делает)

на не очень большой срок

Истории использования

Выброс решения

**Техники в XP**

1) мы пытаемся спланировать операции, на которых настроен пользователь, и, если у нас есть проблема с выполнением плана, то план нужно менять.

2) Частая смена версий, много релизов.

Пользователь быстро получает хоть что-то

3) Использование метафор(несколько емких предложений)

4) Простые решения(не нужно наворотов, делай то, что хочет пользователь)

5) Постоянная переработка для улучшения када

6) Разработка на основе тестов

Сначала делаем тесты, потом код

7) Парное программирование

один – прогает

второй – под руку советует (кто, что делает решают сами динамически)

8) Коллективное владение кодом

то есть нету областей ответственности

**9**

1. Постоянная интеграция – как только появляется новый кусок кода, его нужно вставить в систему.
2. Код – основное средство коммуникации.
3. 40-часовая рабочая неделя, переработка не приветствуется (т. к. означает, что не вошли в график и теперь делается наспех и некачественно).
4. Включение заказчика в систему (всегда должен отвечать на вопросы «Что вы хотите».
5. Открытое рабочее пространство.
6. Изменение правил по необходимости.

## C3-system проект

Задача – сделать систему, которая контролирует все о работниках некоторой компании. Через год стало ясно, что проект не удался. Потом сделали на 10 тыс. человек, но дальнейшего расширения так и не добились. Но именно тогда были написаны многие книги по основам программной инженерии.

# Работа с требованиями

Нужды, потребности, ограничения системы. Также нужно иметь в виду концептуальную бизнес-модель.

Разработанная система решает некоторую проблему => изобретение наборов функций (абстрактных характеристик) для решения проблемы – обычно именно об этом разговаривают с заказчиком («пользовательские требования») => детализация требований (каждую функцию преобразовать в отдельный документ, детально описывающий характеристики).

IEEE-830 – стандарт на требования к ПО.

IEEE-1233 – стандарт на требования к программно-аппаратной системе.

## Свойства IEEE-830

1. Корректность, адекватность
2. Полнота
3. Согласованность (непротиворечивость)
4. Ранжирование по важности
5. Ранжирование по стабильности (т. е. при выполнении требований нужно учитывать вероятность, что они могут измениться)
6. Проверяемость (т. е. всегда должен существовать человек, который может сказать, правильно или неправильно работает система)
7. Прослеживаемость (требования можно привязать к каким-то элементам системы)
8. Удобство внесения изменений (должны оформляться таким образом + они описаны систематически)
9. Однозначность

## Набор деятельностей

(Порядок может быть сдвинут)

1. **Выделение требований**
   1. Нужно выделить все возможные источники (документы / люди (пользователи, эксперты))
   2. Извлечение требований из источников:
      1. Аналитические методы (думаешь сам)
      2. Коммуникационные методы (пристаешь к тем, кто знает):
         1. опросы
         2. демонстрационные
         3. когнитивные (пытаемся максимально отвлечься от предметной области)
   3. Согласование
2. **Описание и систематизация требований**Описываются *варианты использования* – основной сценарий и некоторый набор альтернатив + описание результатов на выходе.  
   *Абстрактные варианты использования* – абстрактные решения задачи.  
   *Детализированные варианты использования* – конкретные сценарии решения задачи.  
     
   *Связи* – по использованию – т. е. один сценарий использует другой сценарий.  
     
   Любой процесс можно описать диаграммой, но более детально.  
     
   **DFD** – диаграммы потоков данных – процессы + потоки данных + хранилища + внешние сущности.  
   **ERD** – сущности + связи + атрибуты.  
   *Сущности* – некоторые понятия, например, товар, заказ, клиент.  
   *Связи* ­– обязательно / не обязательно (один к одному / один ко многим / многие ко многим).  
     
   ISO-9116 – набор характеристик качества по:
   1. функциональности
   2. надежности (как часты ошибки и как дорого устраняются)
   3. производительности
   4. переносимости
   5. удобству использования
   6. удобству сопровождения

Другой способ *системаизации* ­– фото 13, таблица сверху.  
  
Также бывают **позитивные/негативные требования**, **требования при сбоях**.

1. **Валидация и верификация**

# Ошибки в ПО и борьба с ними

Ошибки есть, так как система сложная.

## Основные причины ошибок

1. Неправильное понимание того, что нужно сделать.
2. Придумали неправильное решение.
3. Опечатки и прочее.

## Способы борьбы

1. Много думать, проверять и перепроверять свое понимание.
2. Стандартные методы контроля качества.

*Defect* – ошибка любого рода (любой природы).

*Failure* – сбой (система себя не так повела).

*Fault* – ошибка в коде, которая не всегда себя проявляет.

*Error* – ошибка, происходящая у разработчика в голове.

Чаще всего ошибки группируются в неясных и «темных» местах.

## Методы контроля качества

1. Валидация – проверка, то ли делает система с точки зрения пользователя.
2. Верификация – проверка, делает ли система то, что записано в документации
   1. Экспертиза – человек смотрит в код и ТЗ и сравнивает
   2. Статический анализ – без выполнения кода
   3. Динамический анализ – с выполнением кода
   4. Формальные методы – строгое доказательство корректности математически

## Инспекция по Фагону

Есть два документа, нужно проверить, соответствуют ли они друг другу (первичный и вторичный).

1. Вводятся роли
   1. Ведущий (организатор)
   2. Инспектор (ищет несоответствия)
   3. Автор (отвечает за первичную документацию)
   4. Интерпретатор (отвечает за вторичную документацию)
2. Вводится последовательность ходов
   1. Планирование (проверка готовности документации, набор людей, планирование совещаний и т. д.)
   2. Обзор – ознакомление с первичной документацией (в формате совещания)
   3. Подготовка – инспекторы сами анализируют
   4. Совместная инспекция – совещание, где каждый рассказывает про найденные несоответствия и задает еще вопросы
   5. Исправление
   6. Контроль ­– проверка, что все хорошо

## Статический анализ

1. Находим стандартное место, где делают ошибки (проблема – много ложных срабатываний)
2. Наборы эвристик (накладываются поверх пункта 1)  
   В основном ошибки двух видов:
   1. *false positive* – ложное срабатывание
   2. *false negative* – не нашли ошибку

## Динамический анализ

Ориентированный на особенности выполнения анализа. Самое популярное: тестирование + профилирование.

1. **«Дедуктивная» верификация**  
   Основывается на сборе трасс и построении модели, основанной на булевых выражениях (системных вызовов или данных и т. д.)  
     
   Проблема с циклами (для борьбы с этим в циклах выделяются инварианты, не влияющие на выход из цикла).  
   Другая проблема ­– пользовательский ввод и взаимодействие с другими программами.
2. **Проверка моделей**Работа модели описывается автоматной моделью. Модели бывают ложными и долгими, но зато большая точность + возможность контрпримеров.  
   Еще можно тестировать модель, а потом проверять соответствие ей системы.

## Тестирование

Проверка соответствия требованиям, выполнение при реальной работе системы, на конечном наборе заранее выбранных ситуаций.

Особенности:

1. Тесты собираются по спецификации, и чем она лучше, тем тесты лучше.
2. Проверка при реальной работе системы.
3. При тестировании проверяются заранее выбранные ситуации.

Хочется уметь выбирать тесты, чтобы их минимальное количество максимально проверяло систему.

## Критерий полноты

Тесты нужно выбрать так, чтобы покрывалось все поведение относительно некоторого критерия для системы.

Критерии могут основываться на:

1. Структуре данных
2. Структуре программы
3. Структуре требований
4. Гипотезах об ошибках
5. Мутациях программы

# Архитектура ПО

Функции -> Сценарии <-> Модули / компоненты <-> Сценарии развития.

## Структура системы

Набор модулей и связей между ними.

## Архитектура

Общие правила (концепция).

## Функциональная декомпозиция

Смотрим на функции, которые нужны от системы, и делаем, как можем. Потом пытаемся предугадать изменения системы и переделать структуру так, чтобы потом было проще менять систему.

Когда в системе очень много разнообразных событий, и в будущем их типов станет еще больше, то делают архитектуру в стиле «репозиторий» – суть в том, что существует репозиторий событий, а множество других модулей над ним работают.

От структуры зависят

1. Свойства системы
2. Распределение работ
3. Демонстрация сценария работы (общение с пользователем)
4. Конфигурации (определение того, что все модули друг с другом работают)

## Software Architecture Analysis Module (SAAM)

Берется архитектура и рассматривается с точки зрения дальнейшего расширения.

Этапы:

1. Описать сценарии развития системы
2. Описать сценарии развития архитектуры
3. Оценить сценарии (можно ли внести модификации без добавления модулей)
4. Оценить архитектуру (если есть модуль, который мешается во всех сценариях, то надо что-то делать)

Оценка зависит от использованных сценариев (т. е. фактически от цели использования).

В системах реального времени, где нужно обрабатывать события, часто используется замкнутый цикл управления.

## Широко распространенные стили архитектуры

1. Конвейер
2. Вывод – возврат
   1. Процедурная декомпозиция
   2. Абстрактные типы данных
   3. Многоуровневая система
3. Интерактивная система  
   Model – View – Controller (MVC)  
   Presentation ­– Abstraction – Control (PAC) – нет View контроллеров – то же, что и MVC, но иерархическая  
   Интерфейс состоит из агентов. Пользователь видит Presentation. У каждого агента свои данные.
4. Хранилища
   1. Репозиторий
   2. Blackboard – обычно используется в системах с искусственным интеллектом