Лекции по «Конструированию программного обеспечения

Литература

1. Макконел С. Совершенный код. Практическое руководство по разработке программного обеспечения. – 2016.
2. Орлов С.А., Цилькер Б.Я. Технологии разработки программного обеспечения. – 2012
3. Смирнов А.А. Технологии программирования. – 2011.

Определение

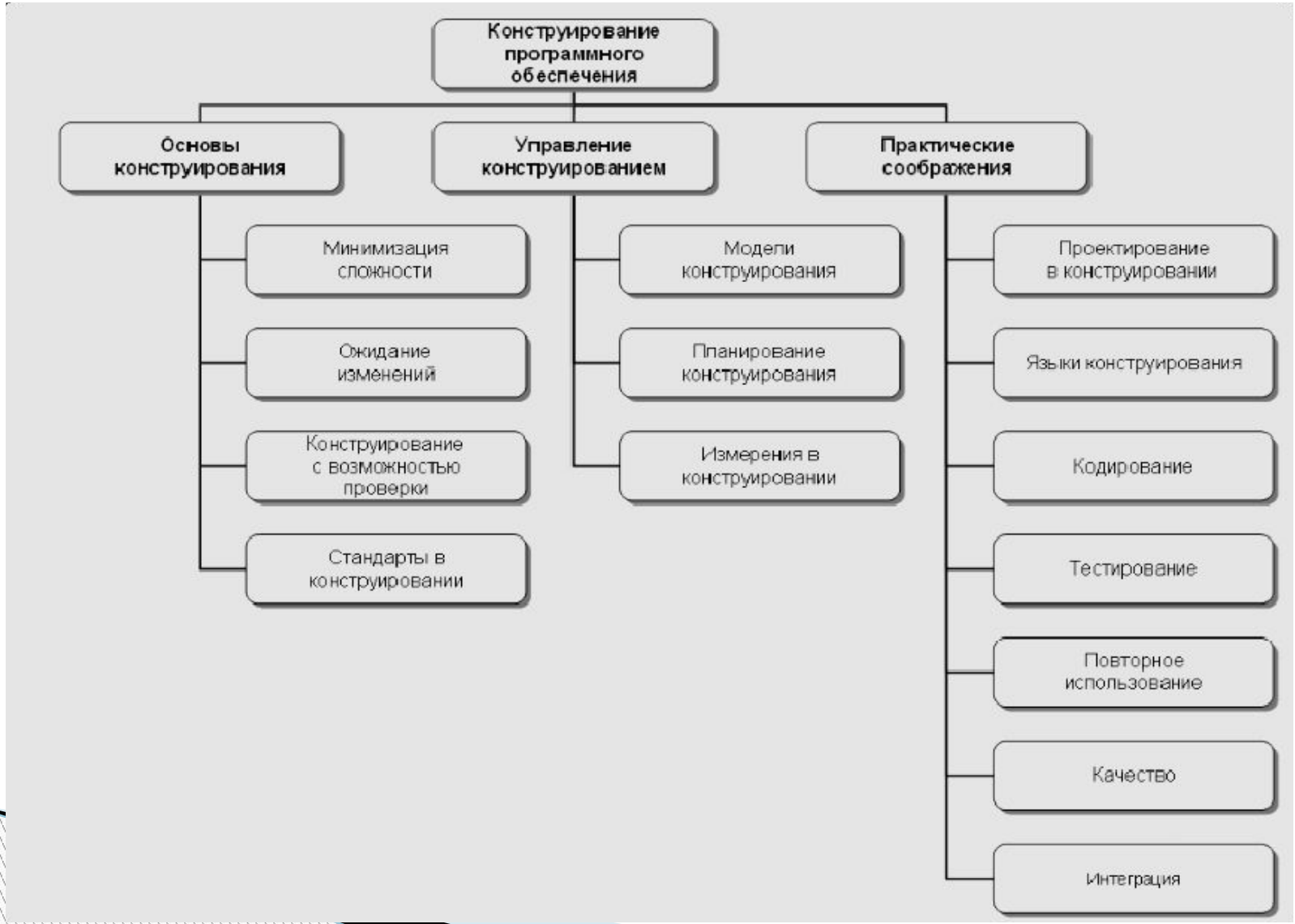
Конструирование программного обеспечения (software construction) представляет собой процесс детального создания программной системы, который раньше называли *программированием*. В рамках этой дисциплины рассматриваются сложные системы, содержащие несколько десятков и сотен тысяч строк и разрабатываемые коллективом программистов.

Этапы разработки программного обеспечения

Основные направления конструирования ПО

1) основы конструирования;
2) управление конструированием;
3) практические аспекты конструирования.

Общая структура предмета



1. Основы конструирования (Software Construction Fundamentals)

1) Минимизация сложности;
2) Ожидание изменений;
3) Конструирование с возможностью проверки;
4) Стандарты в конструировании.

1.1 Минимизация сложности (Minimizing Complexity)

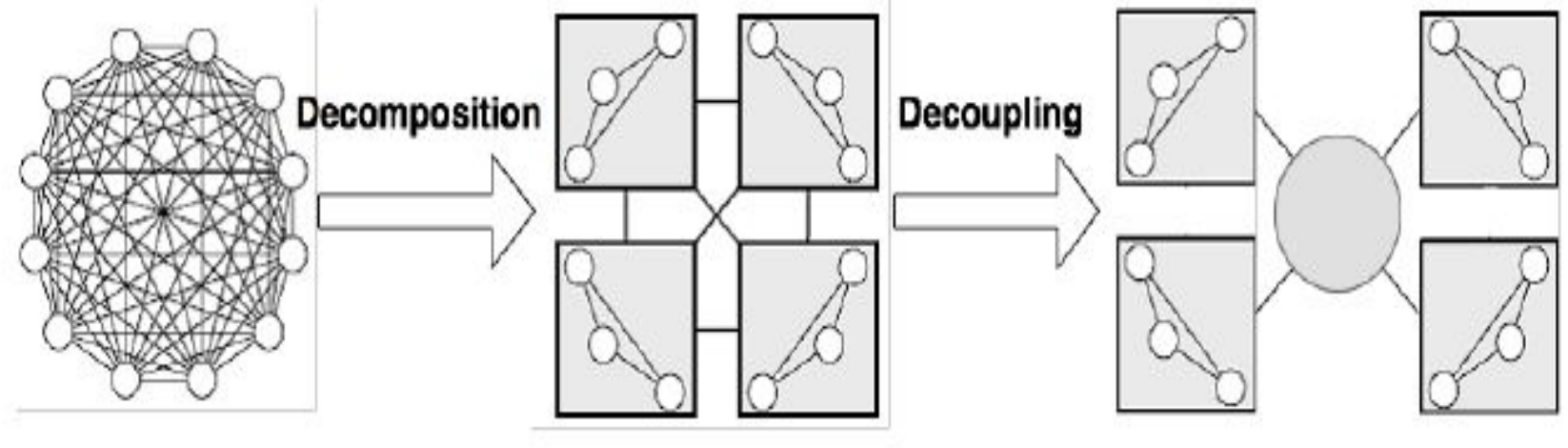
Информационные технологии – отрасль, заставляющая человеческий разум охватывать диапазон информации от отдельных битов до сотен мегабайт (от 1 до 10^9 байт и выше).

Простота достигается с помощью :

1) модульного принципа разработки программ;
2) прототипирования и макетирования;
3) наиболее простых и понятных алгоритмов решения задач;
4) читабельности программного кода;
5) стандартов программирования.

Модульность

Модуль - это функция, процедура или класс, входящие в программную систему. Состав и функции модулей определяются на этапе проектирования системы.



Прототипирование

Прототип – это работающая версия системы, реализующая определенный набор функций. Использование прототипов позволяет демонстрировать заказчику возможности системы и согласовывать с ним ее функции. Оно обеспечивает разработку последовательности версий программного обеспечения.

Простые алгоритмы

облегчают кодирование (написание программы на алгоритмическом языке), чтение программного кода и его отладку. В коллективе для ускорения разработки систем могут применяться методологии парного программирования и обзора программ.

Читабельность

удобство чтения программного кода. Достигается с помощью целого ряда средств: структурирования, использования мнемонических имен и пр.

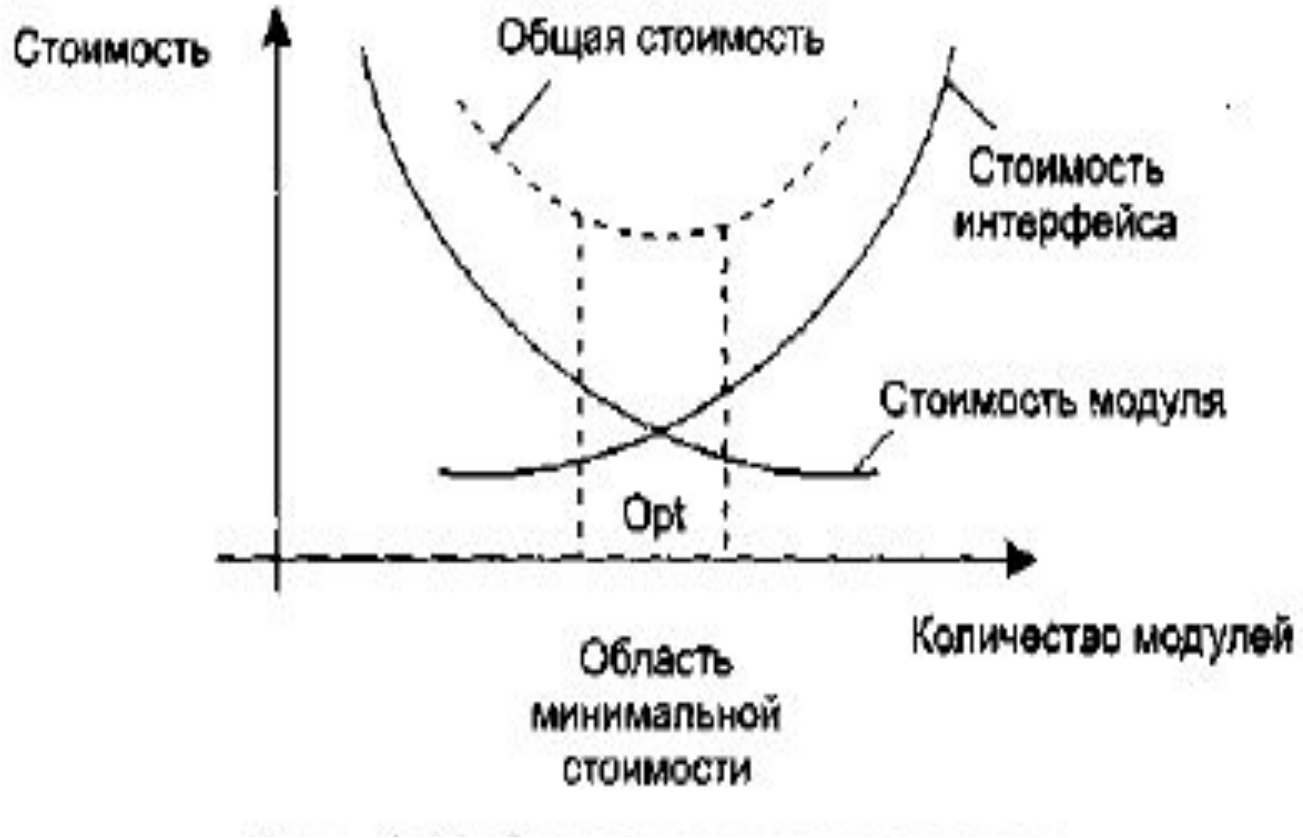
Стандарты программирования

a) общие, разрабатываемые специальными организациями, например, ISO, IEEE или Комитетом по стандартизации РФ,
b) стандарты языка (например, java)
c) а также стандарты конкретной организации-разработчика программной системы.

Модульный принцип разработки ПО

- ***Модуль*** это – фрагмент программы, реализующий один или несколько классов, методов или функций. Обычно он состоит из интерфейсной части и реализации.
- ***Модульность*** – свойство системы подвергаться декомпозиции на ряд связанных между собой частей (модулей). Она обеспечивает интеллектуальную возможность создания сколь угодно сложного программного обеспечения. Это свойство можно проиллюстрировать так.

Затраты на модульность



Свойства модулей

a) информационная закрытость;
b) связность;
c) сцепление.

Информационная закрытость модулей

Связность модулей

Это мера зависимости его частей. Для ее измерения используют понятие *силы связности* (СС).

Типы связности:

1) Функциональная (СС = 10)
2) Информационная (СС = 9);
3) Коммуникативная (СС = 7);
4) Процедурная (СС = 5);
5) Временная (СС = 3);
6) Логическая (СС = 1);
7) По совпадению (СС = 0).

Примеры связанных модулей

1.Функционально связанный модуль содержит элементы, участвующие в решении только одной задачи. Например, вычисление синуса угла, определение координат цели, расчет зарплаты сотрудника и пр.

Приложения, построенные из функционально связанных модулей, проще всего сопровождать.

2. При ***информационной*** (последовательной) ***связности*** элементы-обработчики модуля образуют конвейер (результаты одного обработчика являются входными данными для следующего).

Сцепление модулей

Это мера взаимодействия модулей по данным и измеряется степенью сцепления (
СЦ). Выделяют 6 типов сцепления модулей.

1. *Сцепление по данным* (СЦ = 1), при котором результаты одного модуля являются входными данными для другого, причем каждый параметр является элементарным информационным объектом.
2. *Сцепление по образцу* (СЦ = 3), при котором передаются сложные типы данных.

Сложность программной системы

Показатели:

1) Длина

$$N \approx n_1 \log_2 (n_1) + n_2 \log_2 (n_2)$$

где n_1 - число различных операторов модуля,

n_2 - число различных операндов.

2) Объем

$$V = N * \log_2 (n_1 + n_2).$$

3) Цикломатическая сложность:

$$V(G) = E*N - 2,$$

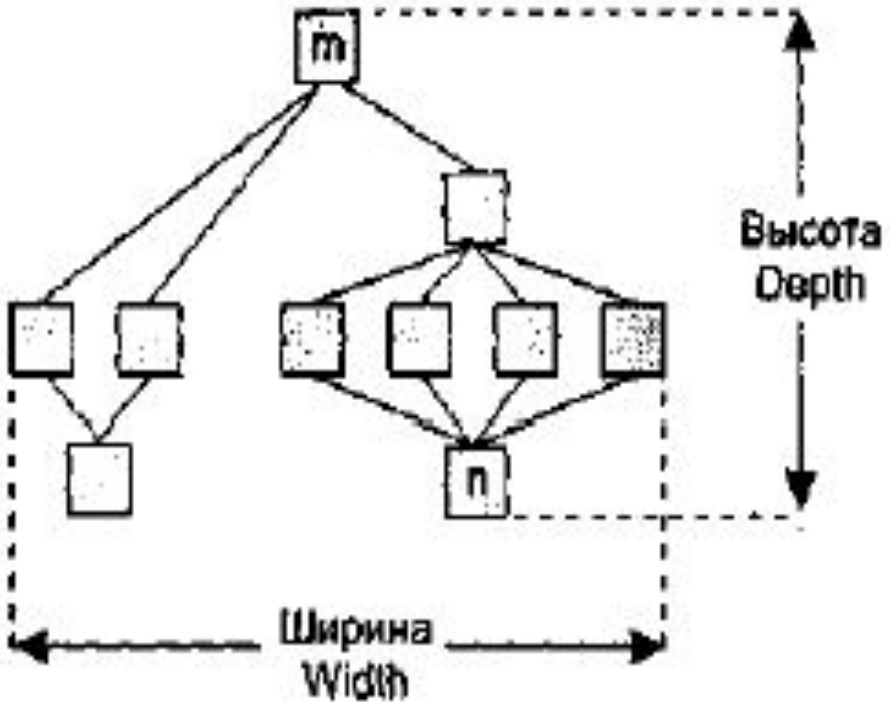
где E – количество дуг, а

N – количество вершин в управляющем графе программной системы.

Характеристики иерархической сложности структуры программной системы

1) Количество вершин (модулей);
2) Количество связей между вершинами;
3) Высота – количество уровней управления;
4) Ширина – максимальное количество модулей, размещенных на отдельных уровнях управления.

Пример иерархической структуры ПС



Высота 4, ширина – 6

Локальные характеристики модулей

a) Коэффициент объединения по входу, Fan_in(i) – количество модулей, которые прямо управляют i-тым;
b) Коэффициент разветвления по выходу, Fan_out(i) - количество модулей, которыми прямо управляет i-тый модуль.

В примере Fan_in(n) = 4, Fan_out(m) = 3.

Прототипирование и макетирование

Это один из этапов разработки программного обеспечения.

Свойства прототипа

1. Этап создания должен быть коротким.
2. Прототипы являются одноразовыми. Они предназначены для того, чтобы донести идею до заказчика. После того как это сделано, прототип может быть отброшен.
3. При его создании, в первую очередь, нужно обращать внимание на важнейшие элементы системы, а также сложные части и их взаимодействие

Этапы разработки прототипа

1. Определение начальных требований.
2. Разработка первого варианта, который содержит только пользовательский интерфейс системы.
3. Изучение прототипа заказчиком и конечным пользователем. Получение обратной связи о необходимых изменениях и дополнениях.
4. Переработка прототипа с учетом полученных замечаний и предложений.

Методологии прототипирования

a) **Быстрое**, при котором макет может быть выброшен, главное - скорость;
b) **Эволюционное**, которое ставит своей целью последовательное создание макетов системы.

Достоинства и недостатки прототипирования

- Достоинства
  a) уменьшение времени, стоимости и рисков;
  b) вовлечение пользователя в процесс разработки
- Недостатки
  a) недостаточный анализ,
  b) смешение прототипа и готовой системы в представлении пользователей,
  c) большое время создания прототипа.

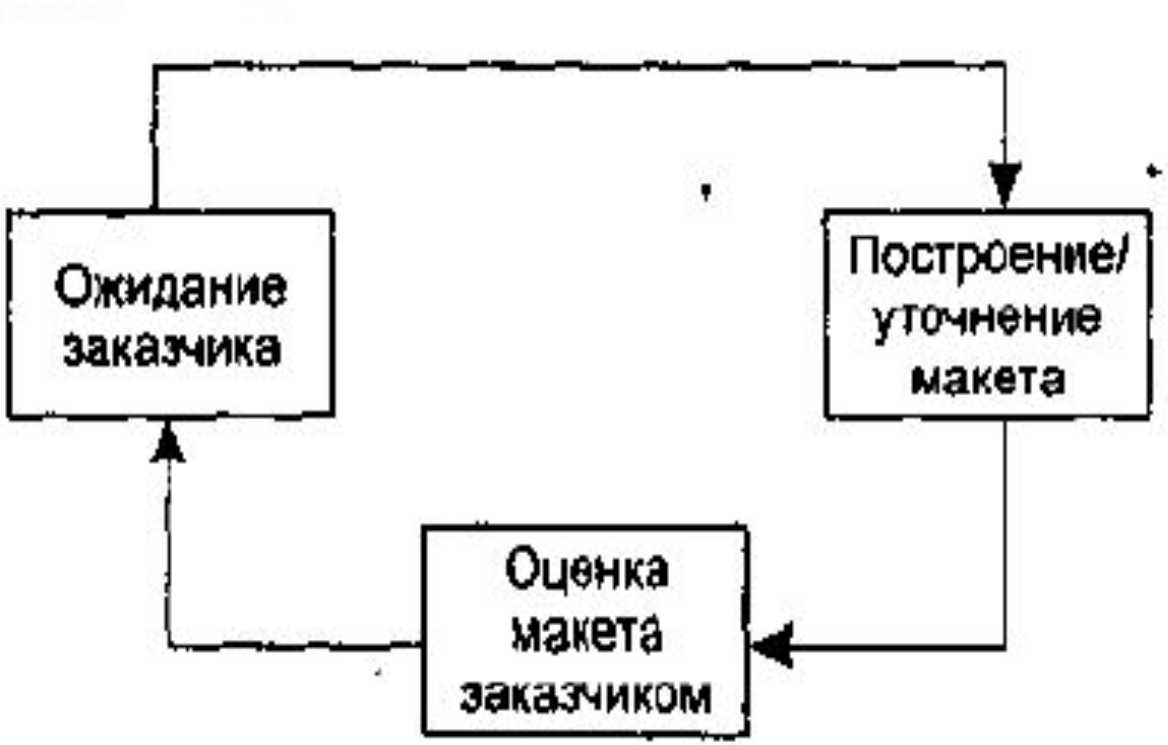
Макетирование

Это процесс создания модели программного продукта. Модель может быть в виде :

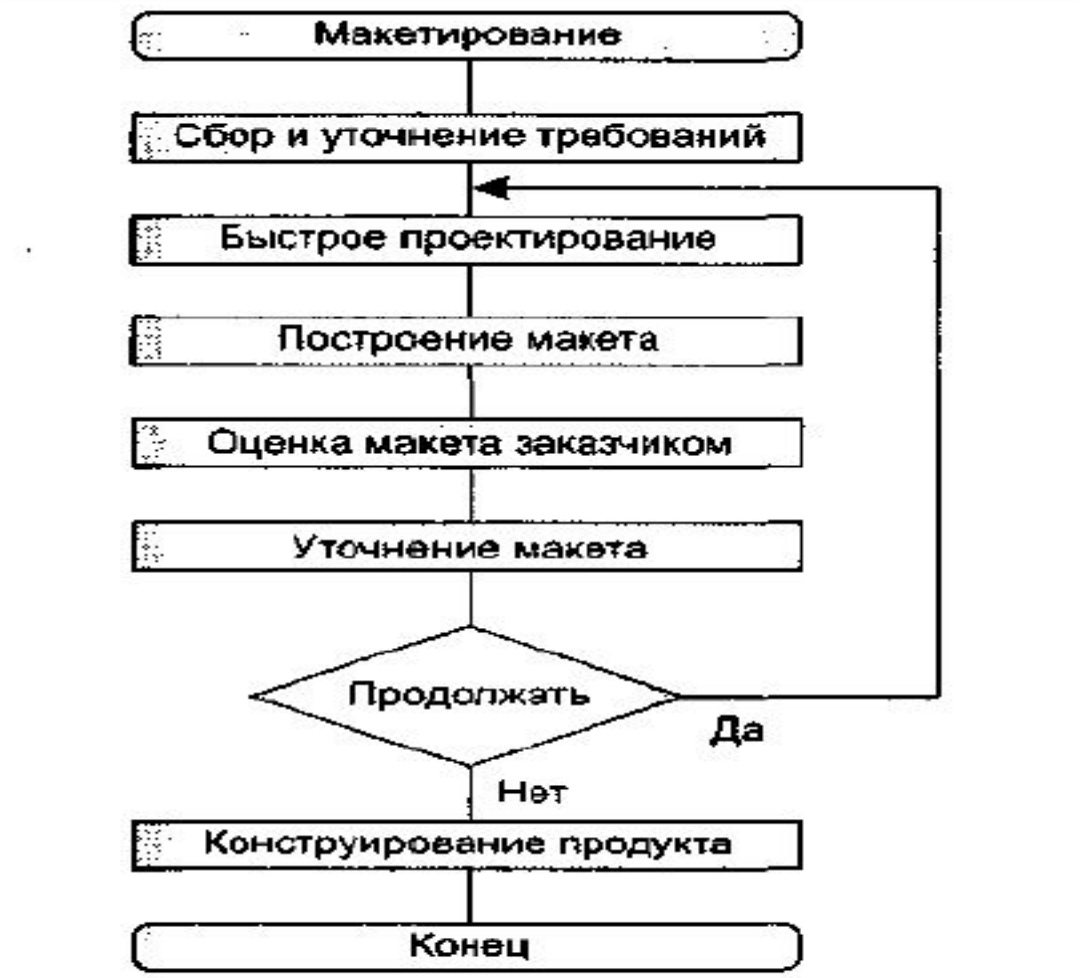
1) Бумажного макета или макета на основе компьютера (изображает или рисует человеко-машинный диалог);
2) Работающего макета (выполняет часть требуемых функций ПС);
3) Существующей программы (характеристики которой затем улучшаются).

Макетирование

Многократное повторение итераций



Последовательность этапов макетирования



Читабельность программного кода

Это свойство текстового материала, характеризующее лёгкость восприятия его человеком.

На читабельность программного кода влияют:

a) Стили отступов = правила форматирования исходного кода,
b) Комментарии,
c) Декомпозиция (разбиение системы на уровни иерархии),
d) Соглашения об именовании данных.

Соглашения об именовании

полезны в следующих случаях.

1) Над проектом работает несколько программистов;
2) Программу будут сопровождать и изменять другие программисты;
3) Программа очень большая, поэтому всю ее один человек не может охватить, а вынужден рассматривать по частям;
4) Программа будет использоваться длительное время, возможно, к ней придется вернуться через несколько недель или месяцев.

1.2. Ожидание изменений

Существует два вида изменений:

1) Усовершенствование кода программы (рефакторинг);
2) Добавление новой функциональности (реинжениринг).

Рефакторинг

Это процесс изменения внутренней структуры программы, не затрагивающий ее внешнего поведения и имеющий целью облегчить понимание ее работы.

Вносятся небольшие изменения, которые легко отследить разработчику.

Применяется постоянно при разработке кода.

Причины необходимости рефактринга

1) дублирование кода;
2) длинный метод;
3) большой класс;
4) длинный список параметров;
5) «жадные» функции — метод, который часто обращается к данным другого объекта;
6) избыточные временные переменные;
7) классы данных;
8) несгруппированные данные.

Реинжиниринг

Это - процесс создания новой функциональности или устранения ошибок путем глобального изменения, но на основе уже имеющегося в эксплуатации программного обеспечения.

1.3. Конструирование с возможностью проверки

Использует следующие техники:

1) обзор и оценка кода;
2) модульное тестирование;
3) структурирование кода совместно с применениям автоматизированных средств тестирования;
4) ограниченное применение сложных или тяжелых для понимания языковых структур.

Обзор кода

заключается в совместном внимательном чтении исходного кода и высказывании рекомендаций по его улучшению.

Считается, что автор кода во время обзора не должен давать объяснений, как работает та или иная часть программы. Алгоритм работы должен быть понятен непосредственно из текста программы и комментариев.

Ошибки в чужом коде замечаются легче.

Недостаток – высокая стоимость.

Модульное или юнит-тестирование

процесс, позволяющий проверить корректность отдельных модулей исходного кода программы.

Задачи модульного тестирования

1. Поиск и документирование несоответствий требованиям
2. Поддержка разработки и рефакторинга низкоуровневой архитектуры системы и межмодульного взаимодействия.
3. Поддержка рефакторинга модулей
4. Поддержка устранения дефектов и отладки

Стандарты в конструировании

1. Государственные (ГОСТ) - ЕСПД;
2. Отраслевые (ОСТ) – стандарты языков (СИ, Java);
3. Стандарты предприятий (СТП);
4. Стандарты проектов – соглашения об оформлении кода и пр. документов.

2. Управление конструированием

2.1 Модели конструирования

Принципы разработки ПО

1. методы,
2. средства и
3. процедуры

Методы **разработки ПО**

обеспечивают решение следующих задач:

a) Планирование и оценка проекта;
b) Анализ системных и программных требований;
c) Проектирование структур данных, алгоритмов и программных структур;
d) Кодирование;
e) Тестирование;
f) Сопровождение.

Средства (утилиты)

обеспечивают автоматизированную или автоматическую поддержку методов. Могут объединяться в системы автоматизированного конструирования ПО. Такие системы называют CASE-системами (Computer Aided Software Engineering).

Процедуры **разработки ПО**

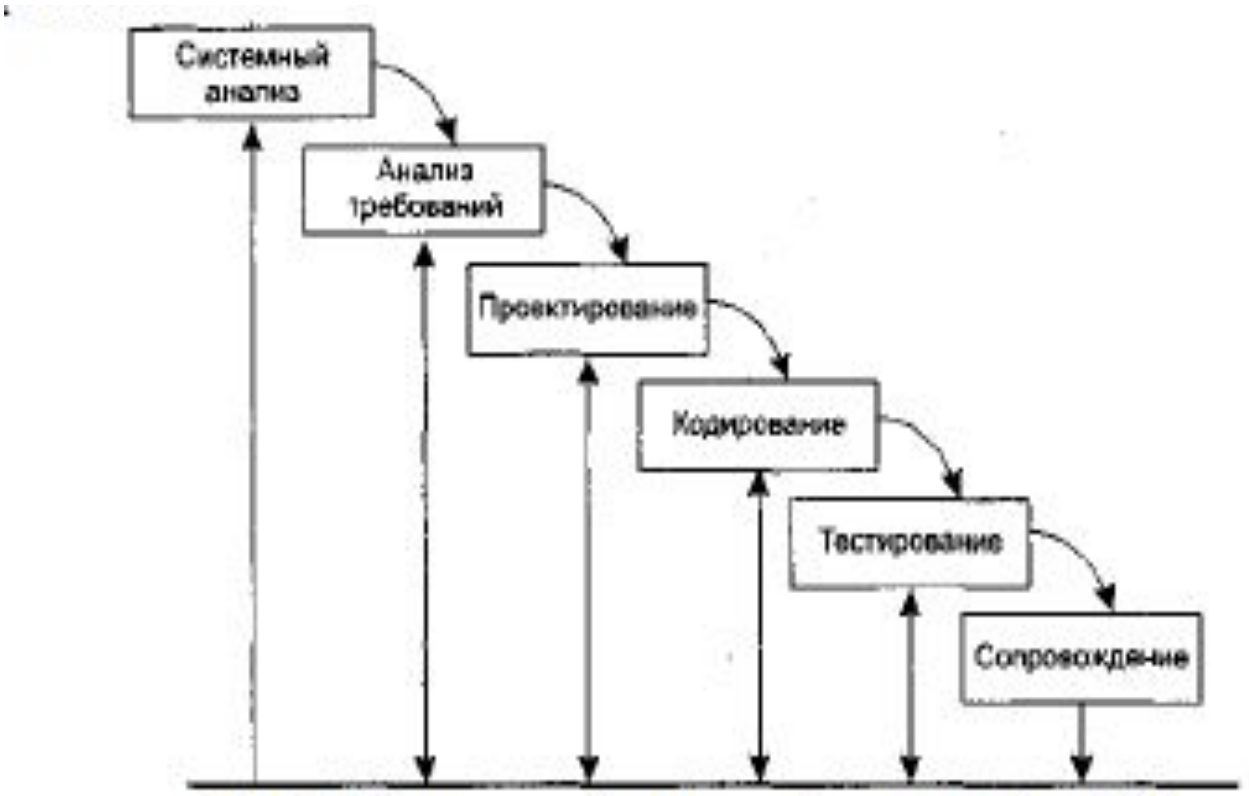
объединяют методы и средства и обеспечивают непрерывную последовательность разработки. Определяют:

a) Порядок применения методов и утилит;
b) Формирование отчетов;
c) Порядок контроля обеспечения качества и координации изменений;
d) Формирование этапов выполнения работ.

Парадигмы и модели жизненного цикла ПО

1) Каскадная или водопадная;
2) Инкрементная;
3) V-образная;
4) Эволюционная (спиральная);
5) Компонентно-ориентированная.

Каскадная модель жизненного цикла ПО



Достоинства и недостатки каскадной модели

Достоинства - упорядоченный процесс конструирования с четким планом и графиком следования этапов.

Недостатки:

a) Требования к проекту на начальном этапе обычно определены частично, поэтому в дальнейшем возможны их уточнения и изменения;
b) Этапы выполняются последовательно, поэтому результаты разработки заказчик получает в самом конце.

Инкрементная модель

объединяет достоинства каскадной и методов макетирования.

V-образная модель

Основным достоинством V-модели является связь разработки программ с их тестированием, валидацией и верификацией.

Спиральная модель

Эволюционная стратегия

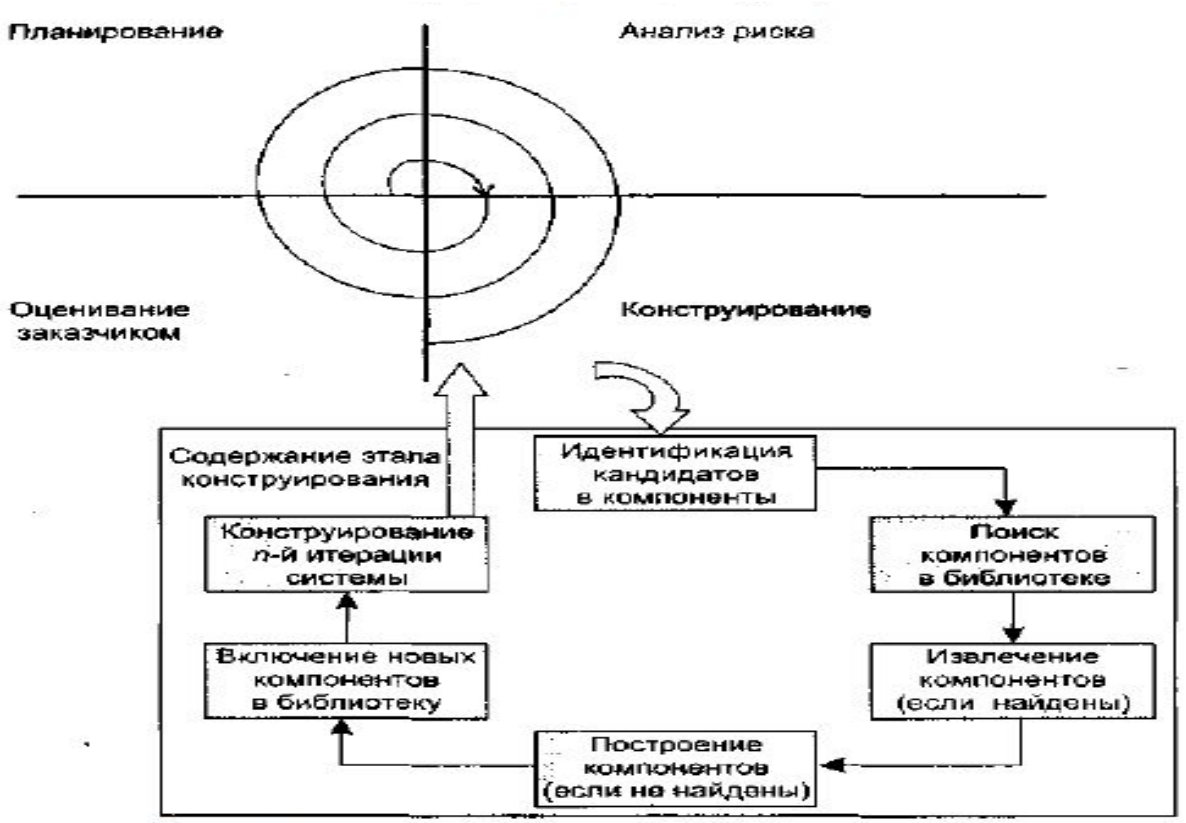
1 – начальный сбор требований и планирование проекта; 2 – та же работа на основе рекомендаций заказчика; 3 – анализ риска на основе начальных требований; 4 - анализ риска на основе рекомендаций заказчика; 5 – переход к комплексной системе; 6 – начальный макет системы; 7 – следующий уровень макета; 8 – сконструированная система; 9 – оценивание заказчиком.

Достоинства и недостатки спиральной модели

- Достоинства :
  a) Более точно отображает процесс разработки ПО;
  b) Позволяет учитывать риск разработки;
  c) Использует моделирование для оценки характеристик.
- Недостатки :
  a) Повышенные требования к заказчику;
  b) Сложность контроля и управления временем разработки.

Компонентно-ориентированная модель

основана на эволюционной стратегии конструирования и является развитием спиральной. Использует библиотеки.



Достоинства компонентно-ориентированной модели

a) Уменьшение времени разработки в среднем на 30%;
b) Сокращение стоимости разработки в среднем на 70%;
c) Увеличение производительности труда в среднем в 1.5 раза.

2.2 Планирование конструирования

Необходимо оценить:

a) людские ресурсы (в человеко-месяцах);
b) продолжительность (в календарных датах);
c) стоимость.

Определяется порядок разработки компонентов и методов обеспечения качества.

Управление инженерией ПО

a) организационное управление (Organizational Management),
b) управление процессом и проектом (Process/Project Management),
c) измерения инженерии ПО (Software Engineering Measurement).

Организационное управление

Задачи :

- управление персоналом (обучение, мотивация и др.),
- коммуникации между сотрудниками и средой (сценарии, встречи, презентации и др.),
- риски (минимизация риска, техники определения риска, выбор решений по их предотвращению и др.).

Управление процессом

- составление плана проекта, построение графиков работ (сетевых или временных диаграмм) исходя из имеющихся ресурсов,
- распределение персонала по работам с учетом сроков и стоимости их выполнения и др.;
- выбор правильной стратегии выполнения плана;
- контроль процесса управления планами и продуктами.

2.3 Измерения в конструировании

Позволяют

- определить или показать качество продукции;
- оценить производительность труда персонала;
- оценить выгоды (прибыль или доход), которые могут быть получены в результате разработки новых программных средств;
- сформировать основу (базовую линию) для последующих оценок;
- получить данные для обоснования запросов на дополнительные средства, обучение и т.п.

Типы измерений

a) Прямые,
b) косвенные.

Метрики программного обеспечения

1) Трудоемкость и емкостная сложность (асимптотическая оценка),
2) Количество строк кода (LOC - lines-of-code),
3) Цикломатическая сложность,
4) Анализ функциональных точек,
5) Количество ошибок на 1000 строк кода,
6) Степень покрытия кода тестированием,
7) Покрытие требований,
8) Количество классов и интерфейсов,
9) Связность.

3. Практические соображения

3.1. Проектирование в конструировании

Нотации проектирования

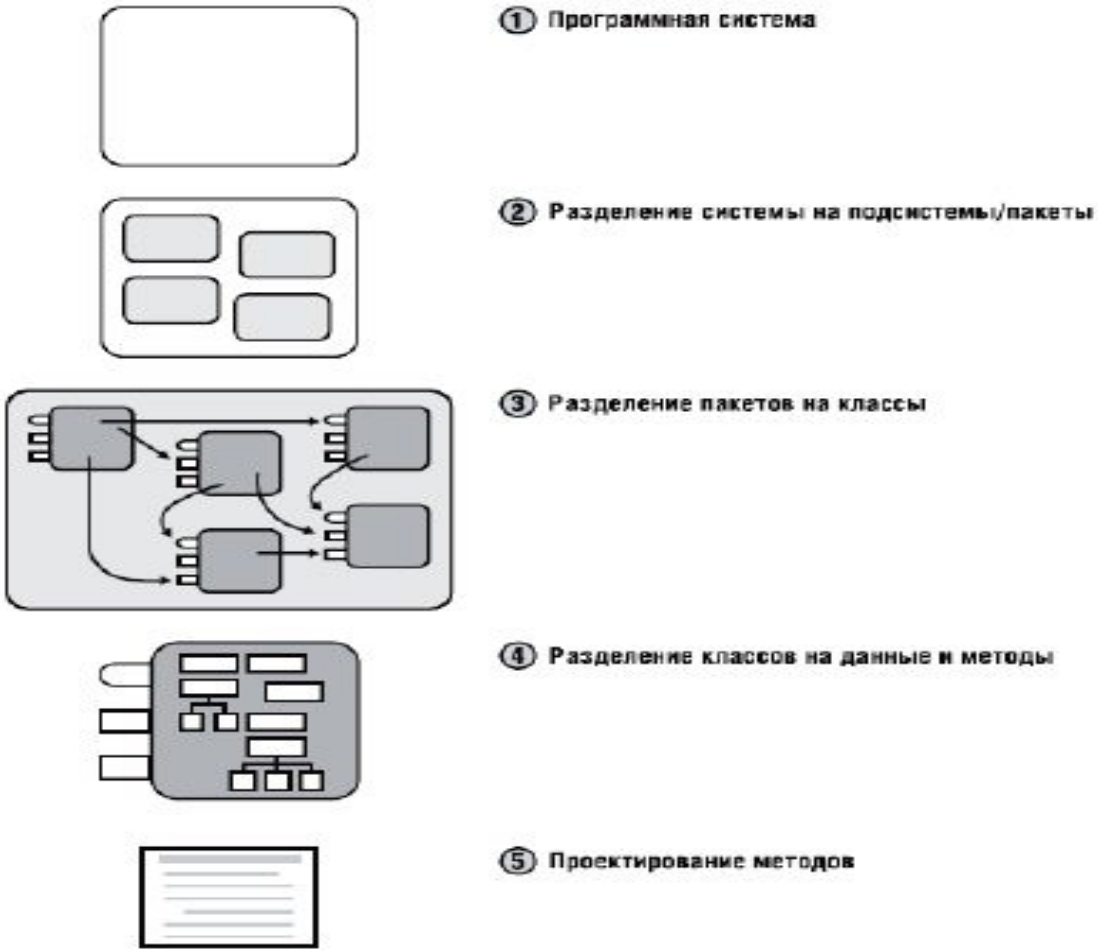
a) Структурные - структурное, схемное или текстовое представление структуры ПО из объектов, компонентов, их интерфейсов и связей,
b) Поведенческие, отражающие динамический аспект поведения систем и их компонентов.

Метафоры (подходы) проектирования

1) Литературная, при которой код пишется как письмо, за столом;
2) Сельскохозяйственная – аналогия с выращиванием растений (каждый блок пишется и отлаживается отдельно);

3) Жемчужины – наращивание функциональности как жемчуг в раковине;
4) Строительная – по аналогии со строительством зданий, предпочтительная.

Уровни проектирования программных систем



Процессы разработки программных систем

1) **Тяжеловесные** – строго и подробно документированные, при которых прогнозируется весь объем работ;
2) **Облегченные** (подвижные - agile) имеют адаптивную природу, требуют меньшего объема документов, ориентированы на человека и учитывают частые изменения требований к программному продукту.

Технологии разработки приложений

1) Быстрая разработка (RAD);
2) Унифицированная разработка (RUP);
3) Экстремальное программирование;
4) Технология Scrum.

Быстрая разработка (Rapid Application Development)

применяет инкрементную стратегию конструирования, обеспечивает очень короткий цикл разработки за счет компонентно-ориентированного конструирования. Эффективна, если требования к системе полностью определены.

RAD-подход ориентирован на разработку информационных систем.

Унифицированная разработка (Rational Unified Process - RUP)

Один из самых известных процессов, использующих итеративную модель разработки. Был разработан компанией Rational Software и стал основной методологией компании IBM.

RUP описывает некоторый абстрактный процесс, на основе которого организация или проектная команда создает специализированный процесс для конкретной системы.

Основные характеристики абстрактного процесса

1) Разработка требований с помощью *прецедентов использования (сценариев);*
2) *Итеративная разработка* с продолжительностью отдельной итерации от 2 до 6 недель.

Жизненный цикл проекта RUP состоит из 4 фаз:

1) Начало - обычно состоит из одной итерации.
2) Проектирование может занимать 2 – 3 итерации или быть пропущенным (если используется уже существующая архитектура). На нем создается архитектура системы. Результатом является 20 – 30 % реализованных прецедентов использования.

Последние фазы

3) Построение длится от 2 до 4 итераций. При этом происходит разработка окончательного продукта.

4) Внедрение занимает от 1 до 3 итераций. На этой стадии проводится тестирование системы, тренинги пользователей и развертывание системы на рабочей площадке.

Экстремальное программирование (eXtreme Programming, XP)

- Облегченный процесс, ориентированный на группы малого и среднего размера (до 10 человек), которые разрабатывают системы в условиях неопределенных или быстро меняющихся требований. XP-процесс – высокодинамичный, состоящий из очень коротких итераций.

Основные действия в XP-цикле

1) Кодирование,
2) Тестирование,
3) Выслушивание заказчика,
4) Проектирование.

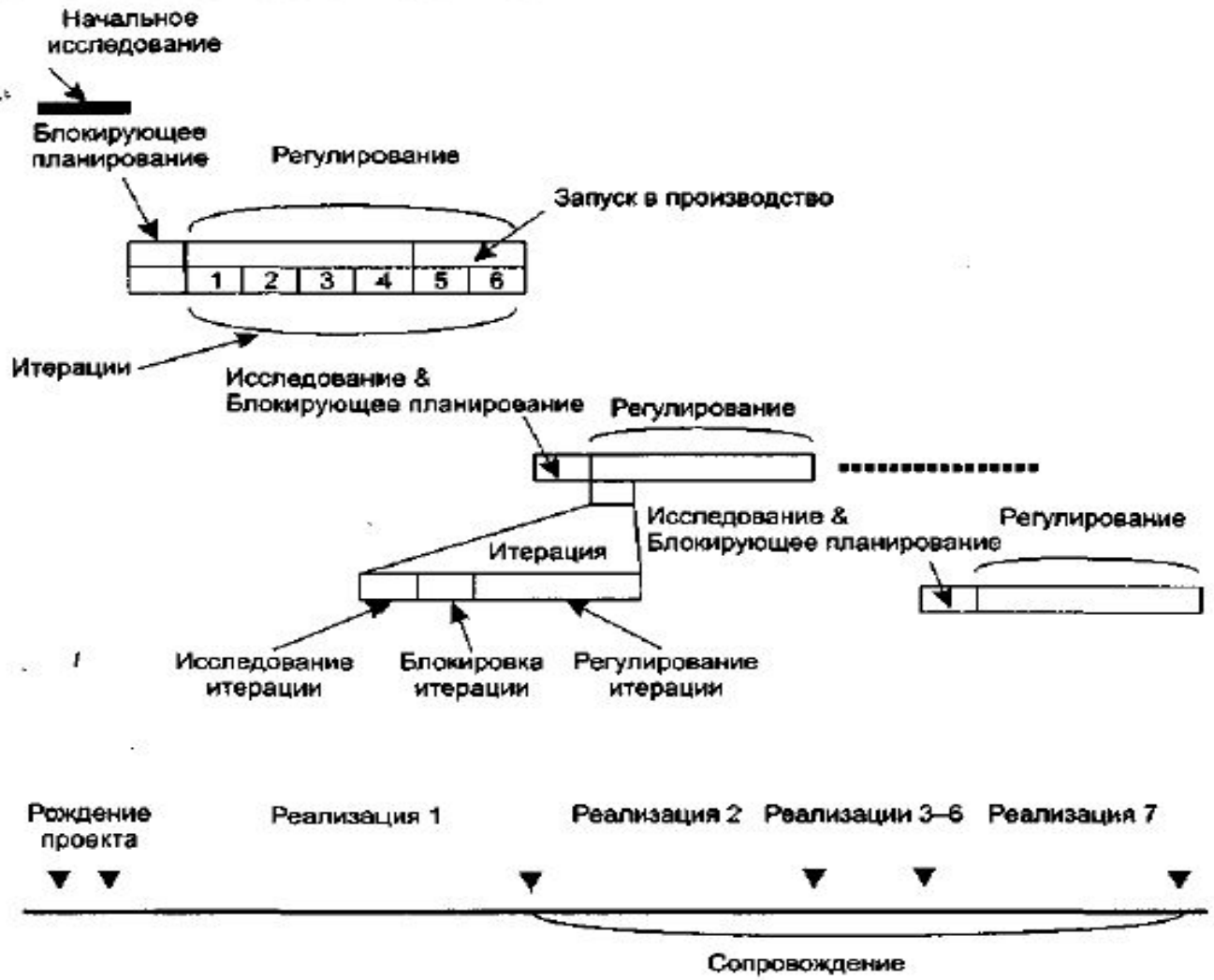
Особенности XP-программирования

Используются 12 базовых методов, которые предполагают разработку историй (сценариев) и включение их в очередную итерацию. Каждая история реализуется за 2 недели.

Применяется парное программирование (написание и отладка кода двумя программистами).

Широко используется тестирование. Тесты составляются до начала кодирования.

Идеальный XP-процесс

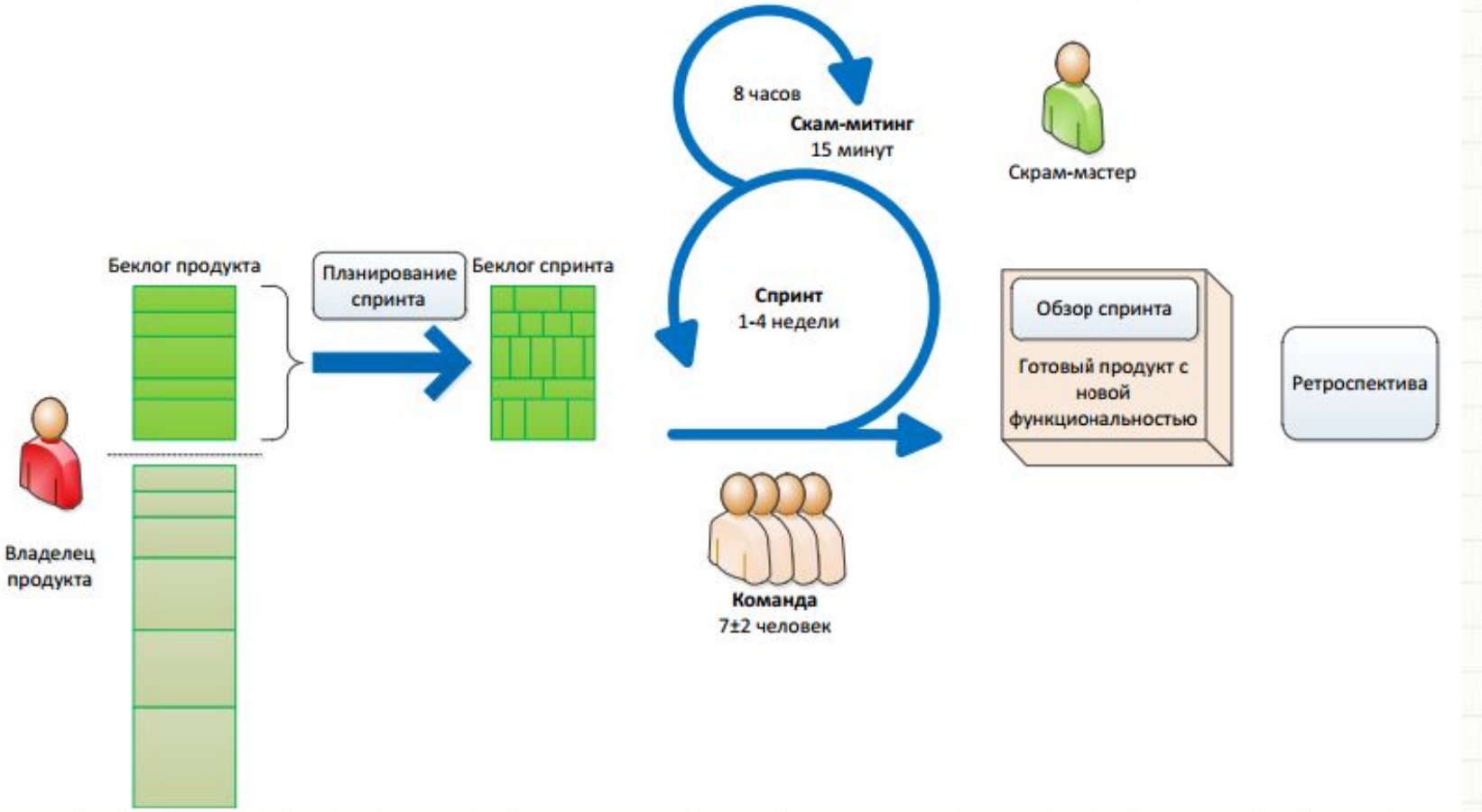


Технология Scrum

представляет собой эмпирический подход к разработке программного обеспечения. Основан на повторяющихся циклах. В процессе разработки участвуют актеры со следующими ролями.

1. *Scrum Мастер* (Scrum Master) – самая важная роль (организует работу).
2. *Владелец продукта* (Product Owner) – представитель заказчика.
3. *Команда* (Team) самоорганизующаяся и самоуправляемая, работает как единое целое, без учета вклада отдельных членов.

Процесс работы над программным продуктом



Спринт

Это короткие ежедневные и циклические 30-дневные встречи.

Отбор задач на спринт выполняется с учетом их важности.

Результатом спринта является продукт, который можно передавать заказчику.

3.2. Языки конструирования программного обеспечения

1) Конфигурационные, которые задают параметры выполнения программной системы;
2) Инструментальные – языки конструирования из повторно-используемых элементов (script);
3) Языки программирования - C++, Java .

Язык программирования

Важную роль играют

1. Программирование не ***на*** языке, а ***с использованием*** языка.
2. ***Опыт*** программирования на конкретном языке.
3. ***Программирование с псевдокодом*** – запись в программе пошагового алгоритма.

3.3 Кодирование

Практика написания программного кода.

1) техники создания легко понимаемого исходного кода на основе использования соглашений об именовании, форматировании и структурировании;
2) использование классов, перечисляемых типов, переменных, именованных констант и других сущностей;
3) организация исходного текста (выражения, шаблоны, классы, пакеты/модули и другие структуры);
4) использование структур управления;

Кодирование

5) обработка ошибочных условий и исключительных ситуаций;
6) документирование кода;
7) тонкая «настройка» кода;
8) рефакторинг.

Качество исходного кода

a) читаемость;
b) лёгкость в поддержке, тестировании, отладке и устранении ошибок, модификации и портировании;
c) экономное использование ресурсов: памяти, процессора, дискового пространства;
d) отсутствие замечаний, выводимых компилятором;
e) отсутствие «мусора» — неиспользуемых переменных, недостижимых блоков кода, ненужных устаревших комментариев и т. д.;
f) адекватная обработка ошибок;
g) переносимость — возможность использования обработчика (компилятора, интерпретатора, транслятора) разных версий или даже различных ОС;
h) возможность интернационализации интерфейса.

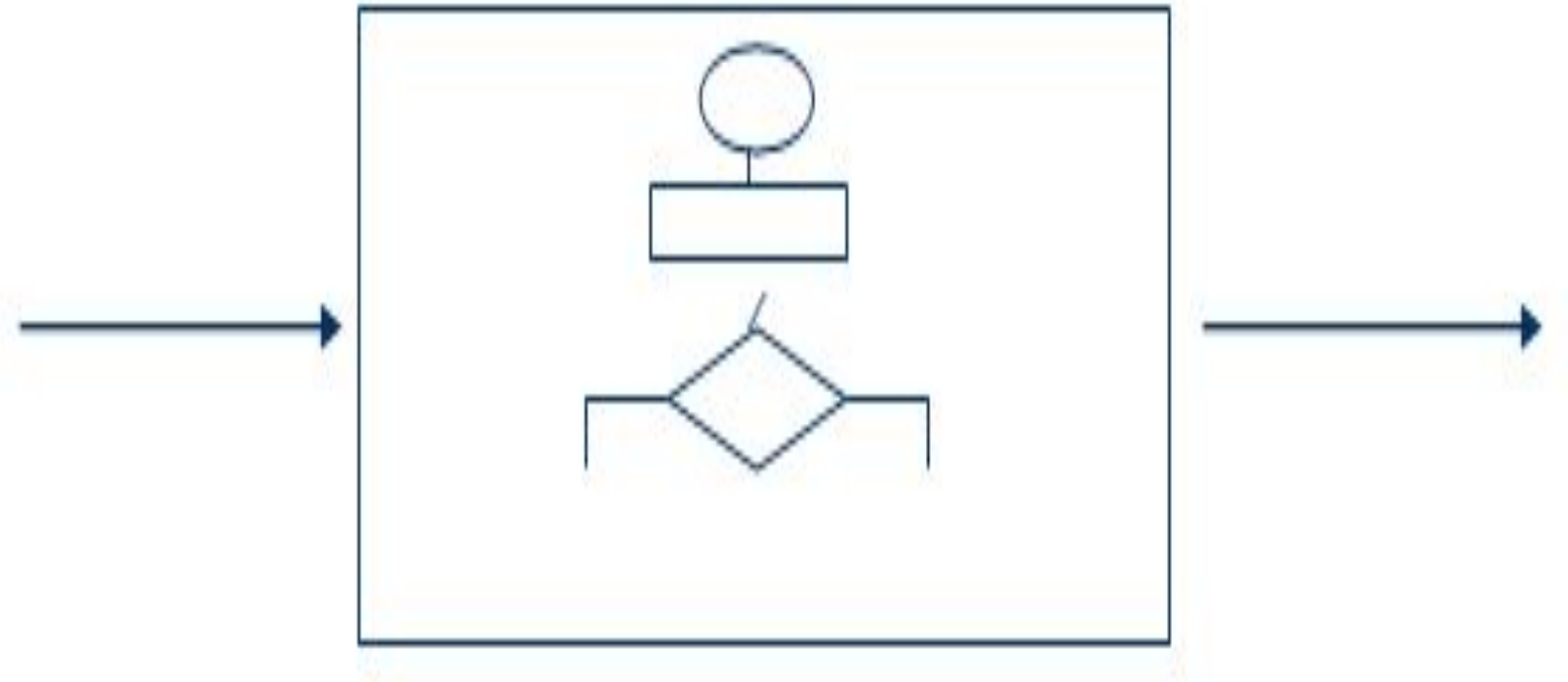
3.4 Тестирование в конструировании

Это процесс выполнения программы с намерением найти ошибки.

Методологии тестирования:

a) «белого ящика» и
b) «чёрного ящика».

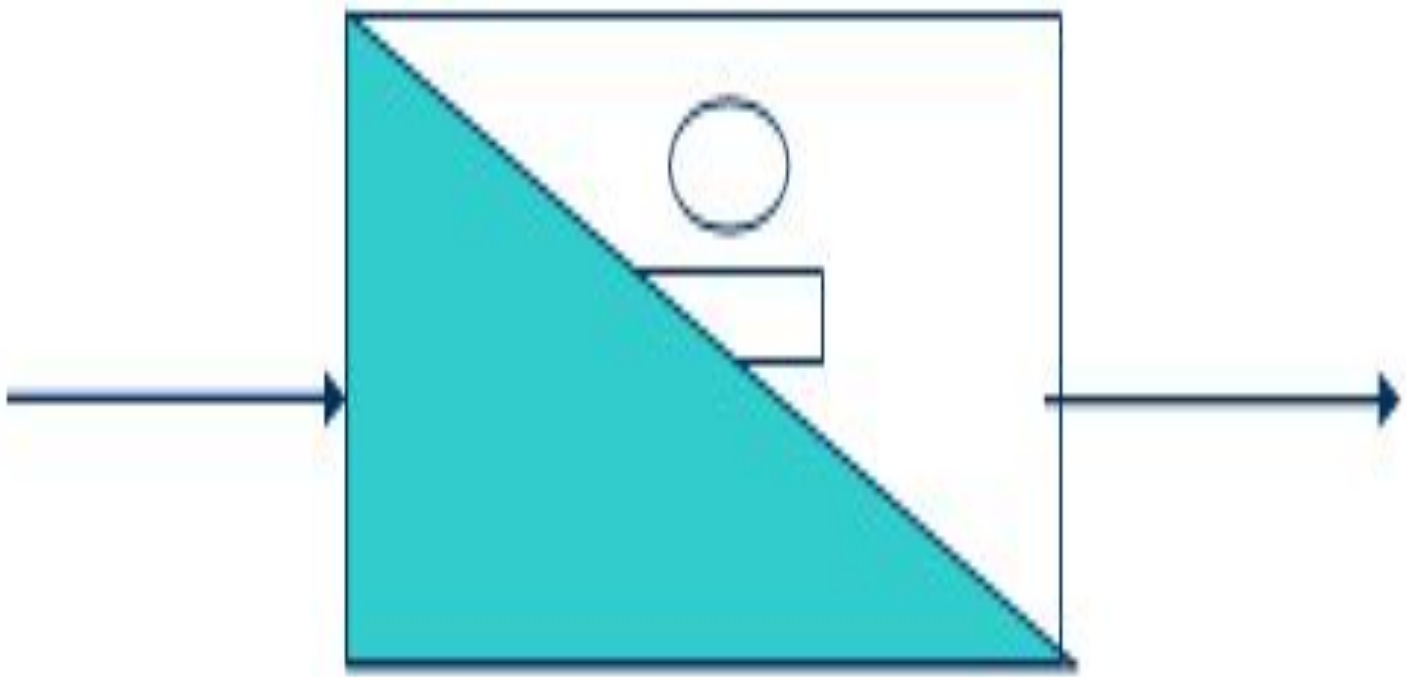
Метод белого ящика



Метод черного ящика

Тестирование внешних интерфейсов

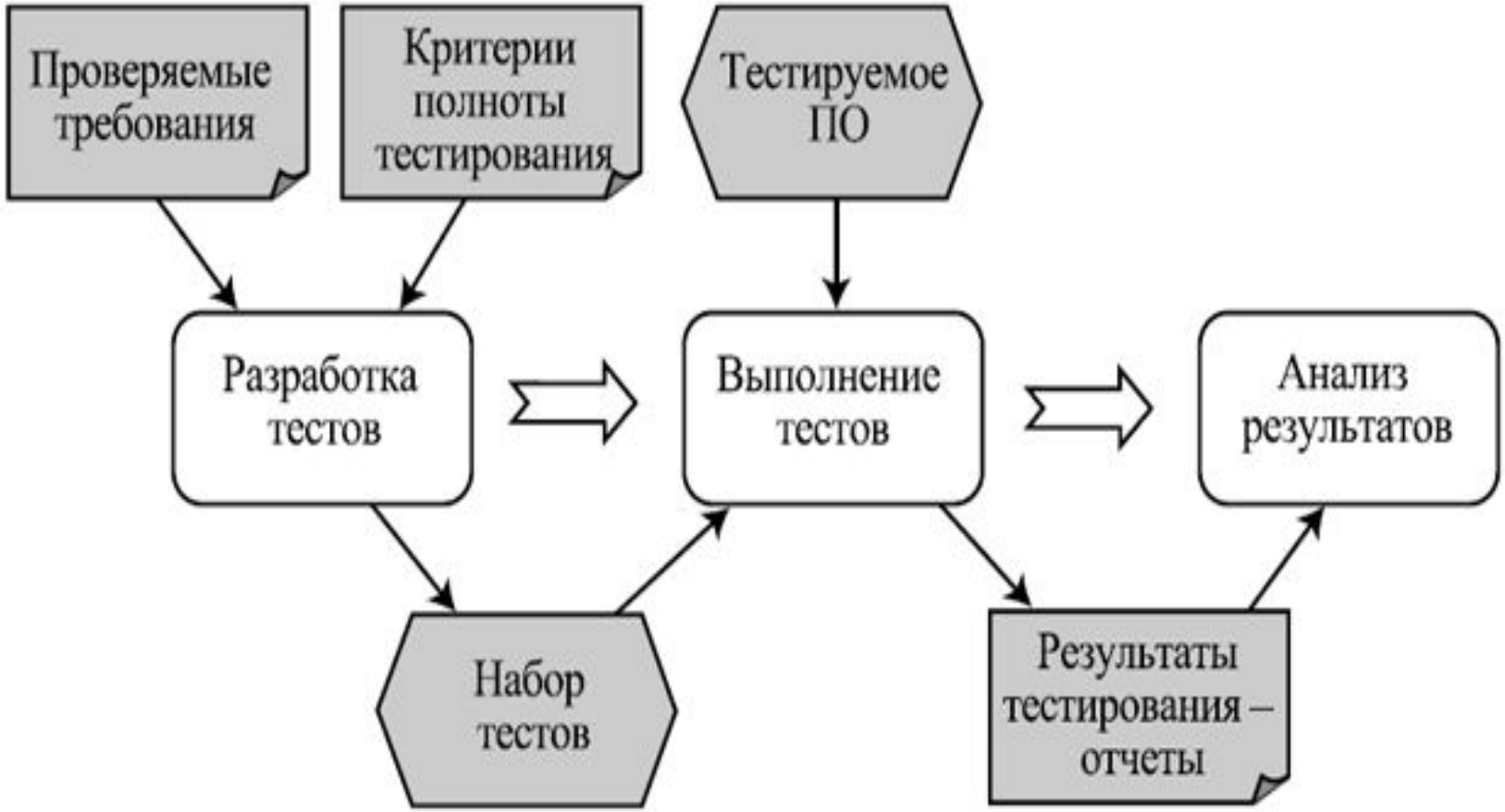
Метод серого ящика



Уровни тестирования

a) Модульное;
b) Интеграционное.

Схема процесса тестирования



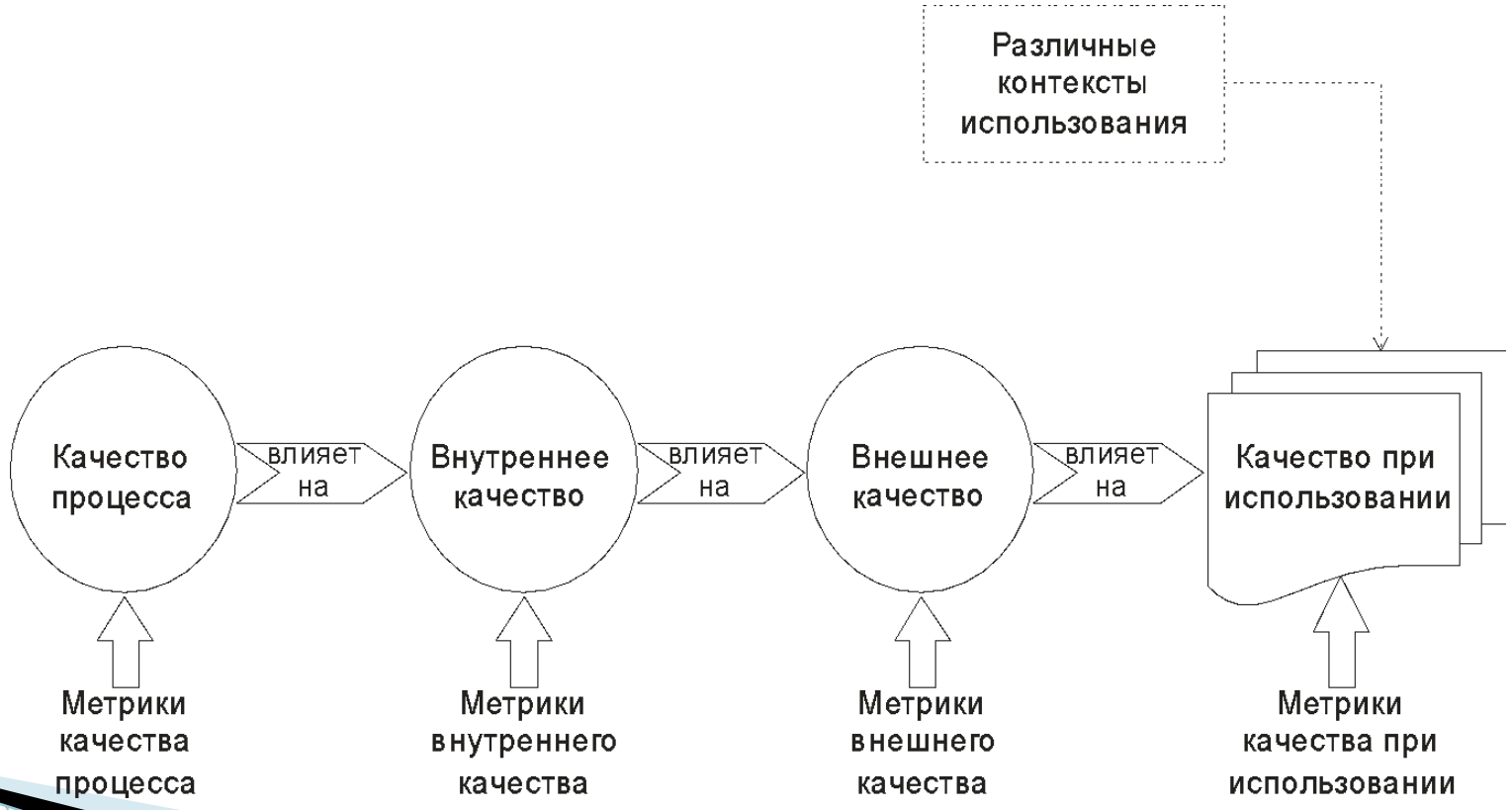
3.5 Повторное использование

Это методология конструирования и проектирования программных систем, заключающаяся в том, что система частично или полностью должна составляться из частей (модулей), написанных раньше, и эти части должны применяться более одного раза (если не в рамках одного проекта, то хотя бы разных).

Применяется для сокращения трудозатрат .

3.6 Качество конструирования

Представление качества в стандарте ISO 9126



Виды качества

- *Внешнее* – качество для заказчика (это удобство в использовании, отсутствие ошибок, хорошая производительность и т.п.)
- *Внутреннее* – это качество для разработчиков программного продукта (соответствие требованиям, удобная архитектура, простота изменения и т. п.)

Характеристики и атрибуты качества ПО по ISO 9126

3.7 Интеграция

Это процесс объединения частей в целое. Наиболее распространенные виды интеграции :

- Объединение модулей в единую программную систему;
- Веб-интеграция — объединение разнородных веб-приложений и систем в единую среду;
- Интеграция данных — объединение данных, находящихся в различных источниках и предоставление их пользователям в унифицированном виде.

Непрерывная интеграция (CI - Continuous Integration)

Это практика разработки программного обеспечения, которая заключается в слиянии рабочих копий в основную ветвь разработки несколько раз в день и выполнении частых автоматизированных сборок проекта для скорейшего выявления и решения интеграционных проблем.

Задачи службы непрерывной интеграции

- получение исходного кода из репозитория;
- сборка проекта;
- выполнение тестов;
- развертывание готового проекта;
- отправка отчетов.

Преимущества непрерывной интеграции

- проблемы интеграции выявляются и исправляются быстро, что обеспечивает минимум затрат;
- немедленный прогон модульных тестов для свежих изменений;
- постоянное наличие текущей стабильной версии;
- немедленный эффект от неполного или неработающего кода приучает разработчиков к работе в итеративном режиме с более коротким циклом.

Недостатки непрерывной интеграции

- дополнительные затраты на поддержку ее работы;
- необходимость в выделенном сервере под ее нужды;
- немедленный эффект от неполного или неработающего кода отучает разработчиков от выполнения периодических резервных включений кода в репозиторий.