Программная инженерия

Этапы жизненного цикла ПО
Зрелость процесса разработки

Этапы жизненного цикла ПО

Этапы разработки ПО

Нету «Православного» деления на этапы разработки ПО:

Sommerville:

- Постановка задачи
- Разработка
- Валидация
- Развитие и поддержка

Unified Process:

- Начало
- Уточнение
- Построение
- Внедрение

Спольски:

- Требования
- Архитектура
- Конструирование
- Тестирование
- Внедрение

Относительная стоимость исправления ошибки

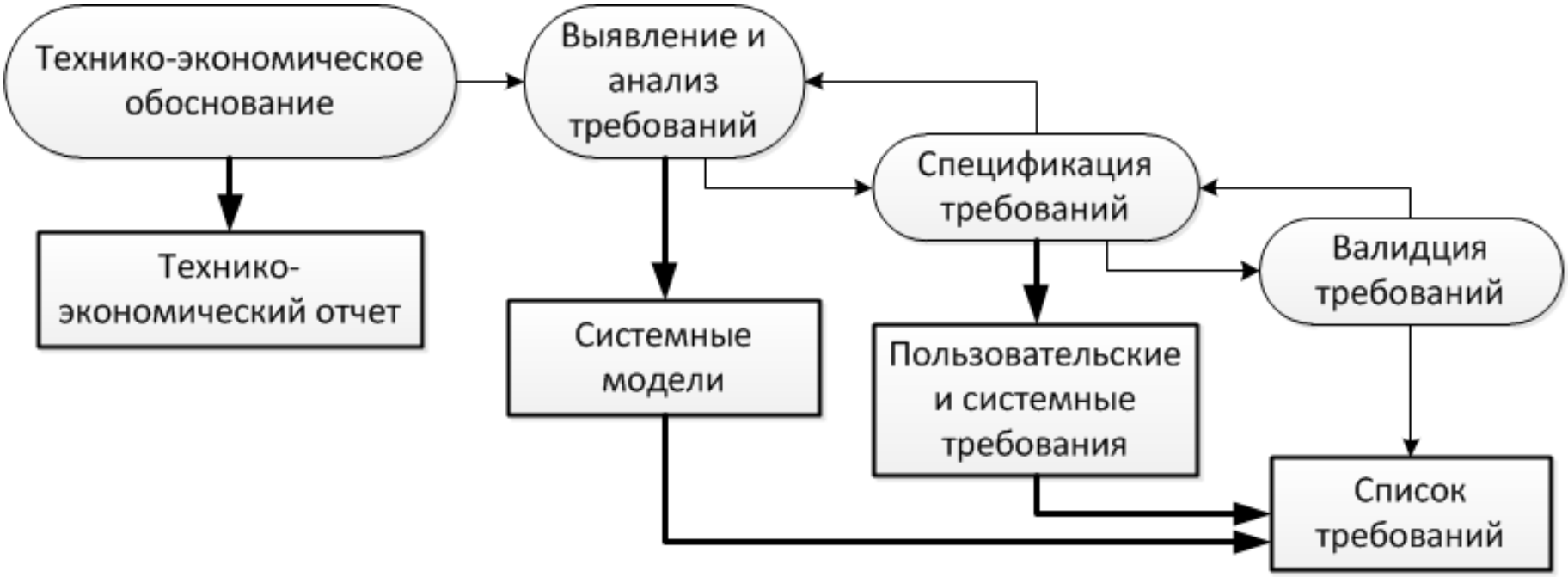
1 Постановка задачи

- *Постановка задачи* – это процесс определения **набора сервисов**, которые должна предоставлять система, а также определения **ограничений**, в рамках которых система будет разрабатываться и исполняться.

1 Фазы постановки задачи

1. Технико-экономическое обоснование
2. Выявление и анализ требований
3. Спецификация требований
4. Валидация требований

1 Процесс постановки задачи



2 Процесс разработки ПО

- Процесс разработки ПО состоит из двух действий:
  - **Проектирование** – описание структуры ПО, моделей данных, интерфейсов компонентов, алгоритмов;
  - **Реализация** – преобразование спецификации разрабатываемой системы в готовый программный продукт.

2 Модель процесса проектирования

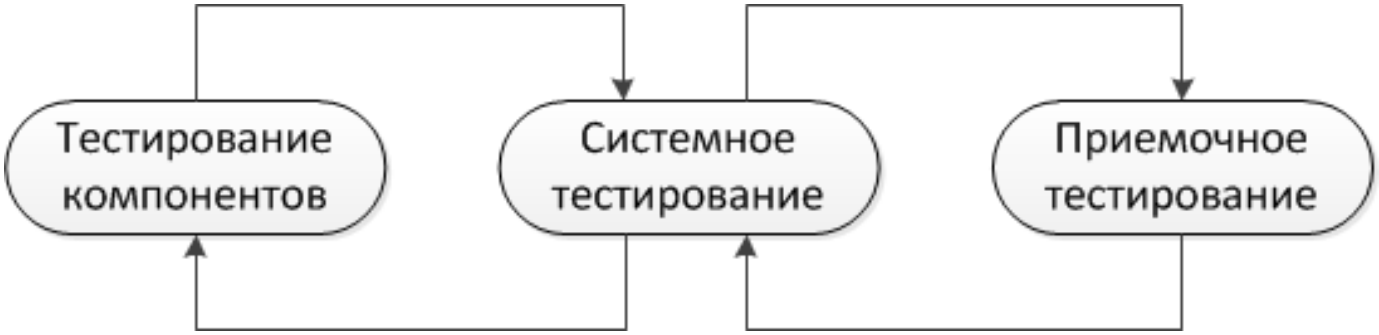
2 Реализация

- Процесс программирования – это индивидуальная деятельность, которая не может быть описана определенным процессом.
- Не смотря на это, в компании могут быть выработаны стандарты кодирования:
  - именование переменных и методов;
  - форматирование исходного кода; методы комментирования и документирования; процесс управления версиями и т.п.)

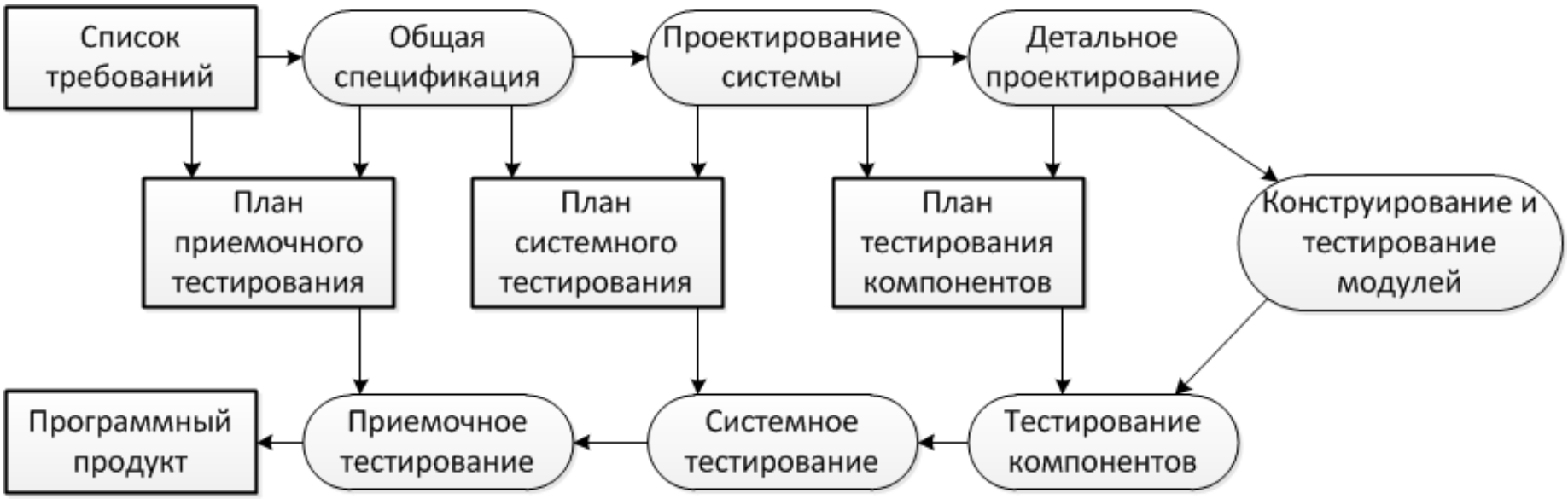
3 Процесс валидации

- Процесс валидации (верификации и валидации) должен показать, что разработанная система соответствует спецификации и желаниям пользователей системы.
- Основной объем затрат в процессе валидации приходится на тестирование системы

3 Процесс тестирования



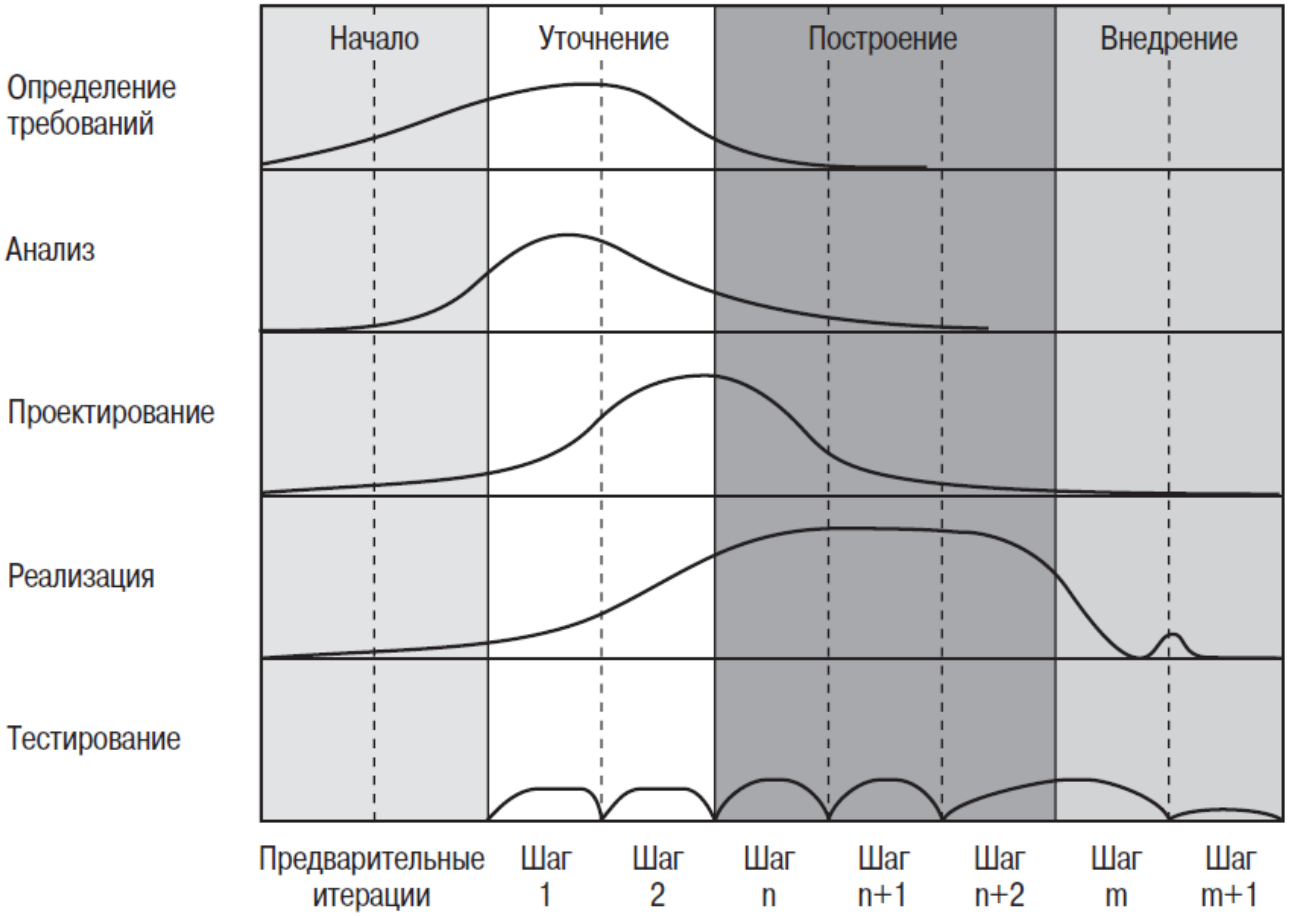
3 Внедрение тестирования в процесс разработки ПО



4 Развитие и поддержка ПО

- Важное отличие ПО от аппаратных платформ и других объектов реального мира – относительная простота изменения и развития
- В связи с этим, процент «абсолютно новых» программных систем очень невысок. Большинство – развитие предыдущих, уже вышедших программных решений.

Действия на этапах разработки ПО в итеративном подходе

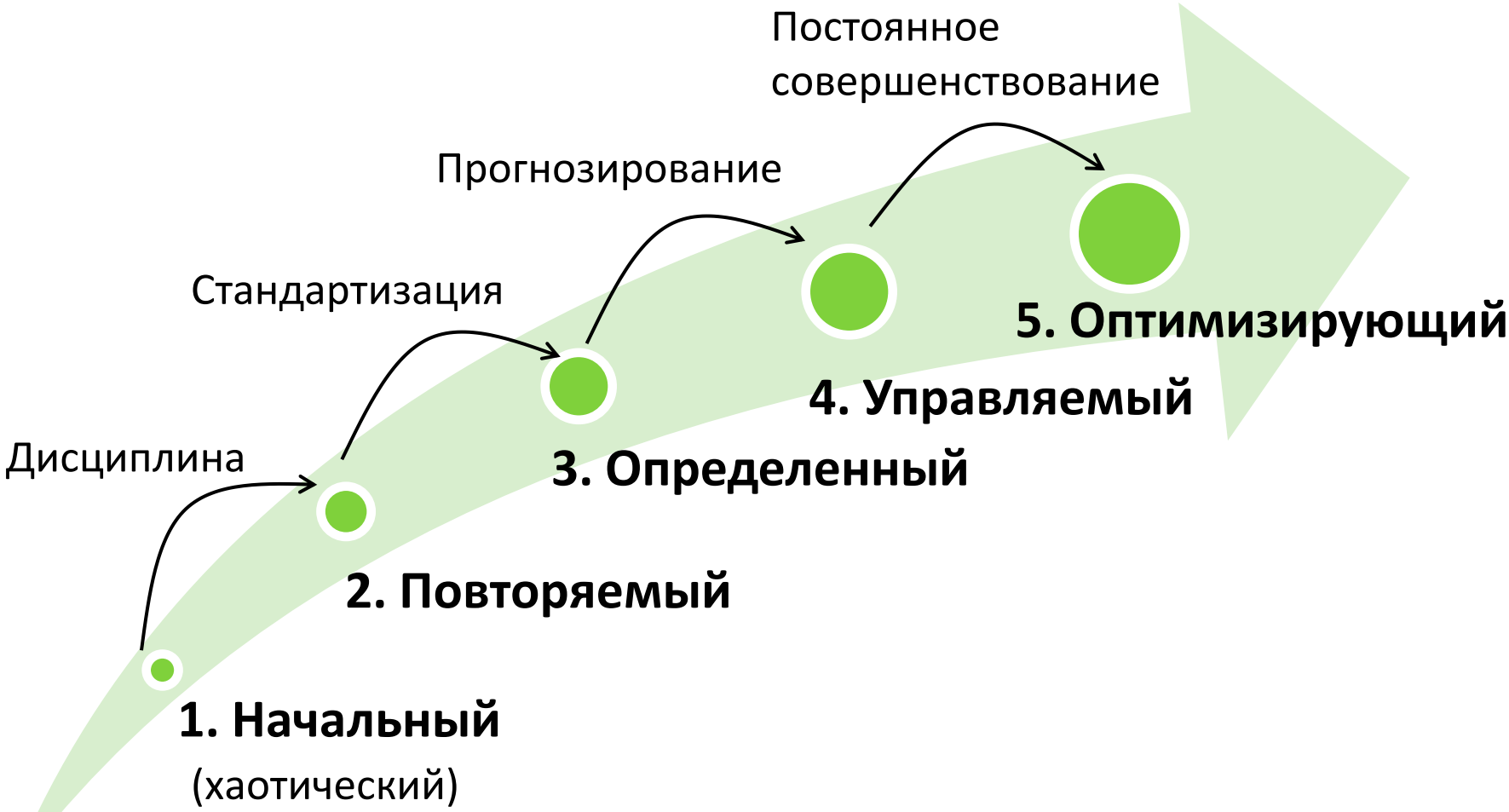


Модель зрелости процессов разработки ПО

Модель зрелости разработки ПО

- Модель зрелости процессов разработки программного обеспечения (Capability Maturity Model for Software, CMM)
- Разработана в **1980** году в Институте программной инженерии при Университете Карнеги-Меллона,
- Де-факто – стандарт уровня зрелости процесса разработки в индустрии производства программного обеспечения.

5 уровней зрелости процесса разработки



1. Начальный уровень

- Отсутствует культура управления
- Неэффективное планирования и плохая работа систем согласования не дает развиться преимуществам хороших решений
- Аврал приводит к полному исчезновению управления, переходу на написание кода и тестирование
- Продуктивность производственного процесса непредсказуема (процесс специально создается для каждого проекта)

2. Повторяемый уровень

- Установлены политики управления проектом разработки и процедуры их применения
- Применяются основные средства управления проектом:
  - Отслеживаются производственные затраты, выполнение графиков и функциональность продукта;
  - Проблемы управления решаются по мере их возникновения;
  - Требования к ПО отслеживаются в системе управления конфигурацией
  - Определены стандарты разработки

3. Определенный уровень

- Процесс разработки и сопровождения ПО надежно документирован, включая как процессы программной инженерии, так и управления («*Стандартный производственный процесс организации (СППО)*» в терминах CMM)
- Реализована общая для организации программа обучения
- Адаптация СППО с целью разработки «*производственного процесса*», учитывающего уникальные характеристики отдельного проекта.

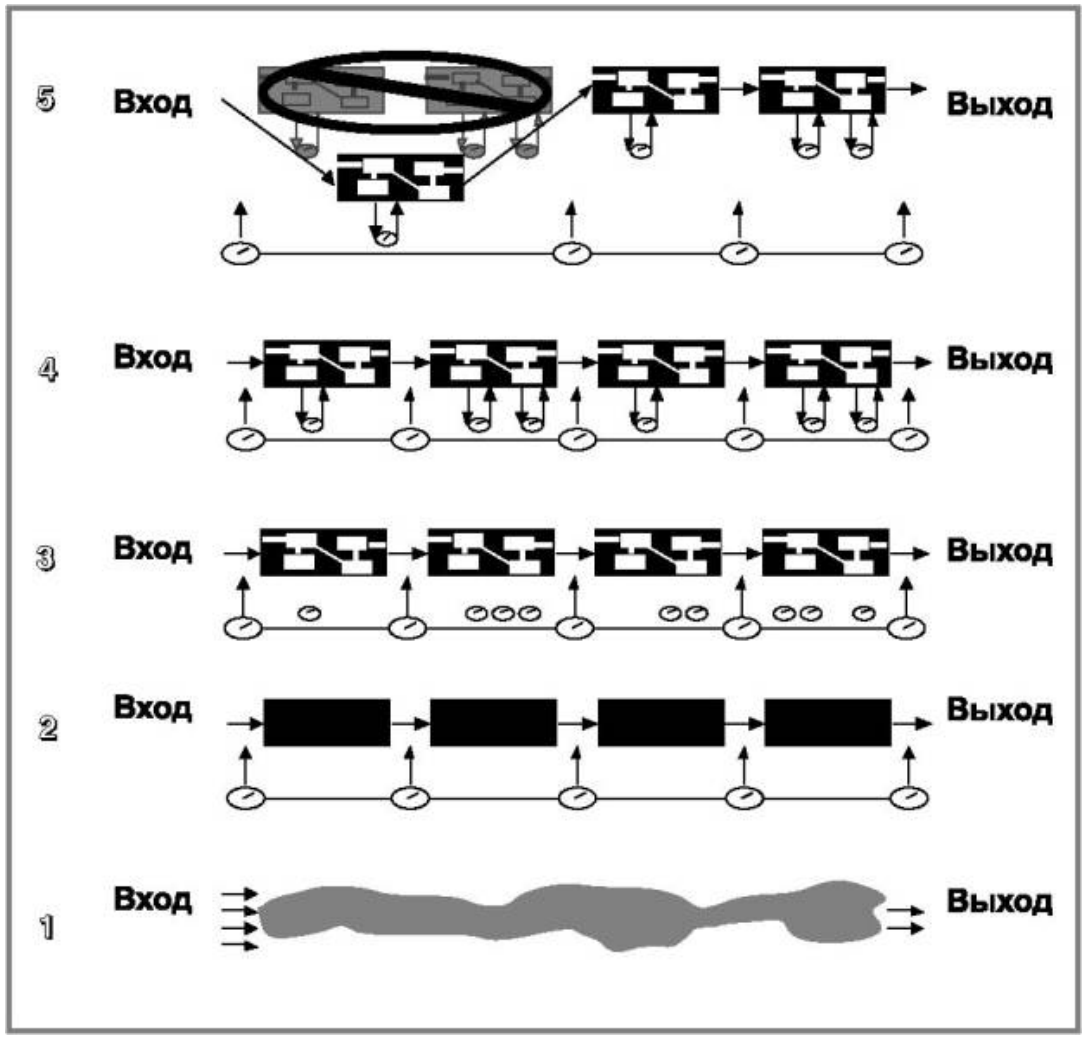
4. Управляемый уровень

- Количественные показатели качества как для программных продуктов, так и для процессов их разработки; измерение продуктивности и качества.
- Производственные процессы оснащены инструментальными средствами для проведения точно определенных и согласованных измерений.
- Позволяет организации прогнозировать тенденции развития процесса и качества продукта

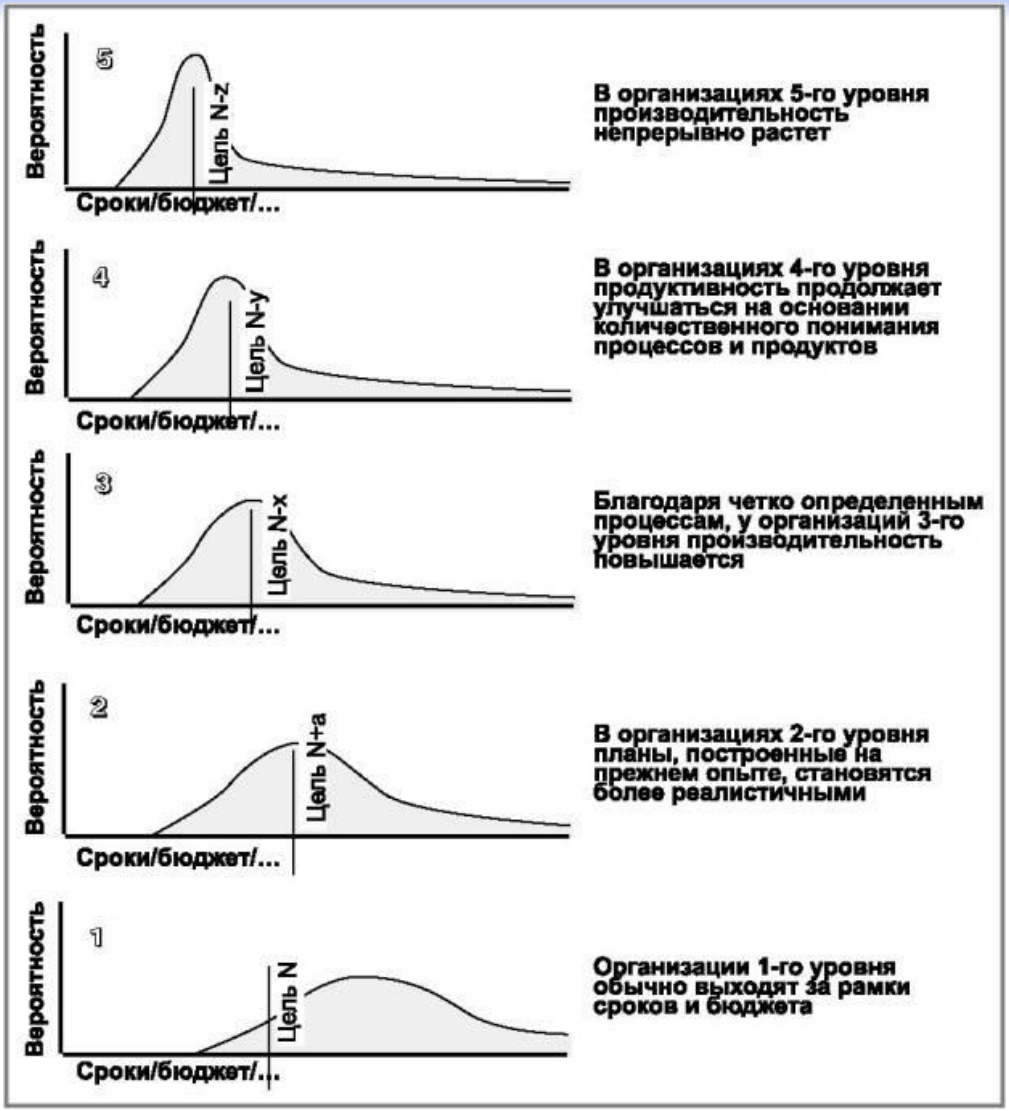
5. Оптимизирующий уровень

- Вся организация полностью сосредоточена на непрерывном усовершенствовании производственного процесса
- Данные по эффективности производственного процесса используются для выполнения стоимостного анализа новых технологий
- Анализ дефектов и определение причин их возникновения. Предотвращение повторения известных типов дефектов в других проектах.

Представление о производственном процессе



Прогнозирование производительности



Развитие в соответствии с моделью CMM

- Каждый уровень образует основу для последующих
- Однако организации могут использовать процессы, описанные на уровнях зрелости, более высоких в сравнении с достигнутыми (анализ требований, проектирование, кодирование и тестирование не обсуждаются вплоть до уровня 3 CMM)
- Пропуск уровней зрелости нежелателен

Ключевые процессы

- Каждый уровень зрелости, кроме первого, состоит из нескольких групп ключевых процессов, указывающих на области концентрации усилий по совершенствованию производственного процесса организации.
- Их можно рассматривать, как требования для достижения уровня зрелости.

Ключевые процессы: Уровень 2

Создание основных средств управления проектом:

1. Управление требованиями
2. Планирование проекта
3. Отслеживание хода проекта и контроль над ним
4. Обеспечение качества ПО
5. Управление конфигурацией ПО

Ключевые процессы: Уровень 3

Как вопросы управления проектом , так и организации в целом:

1. Определение производственного процесса организации
2. Программа обучения
3. Интегрированное управление разработкой ПО
4. Инженерия разработки программного продукта
5. Межгрупповая координация
6. Экспертные оценки

Ключевые процессы: Уровень 4

Установление количественного понимания производственного процесса и создаваемых промежуточных программных продуктов:

1. **Количественное управление процессом** – выявление и коррекция особых причин отклонений внутри стабильного (в целом) процесса.
2. **Управление качеством ПО** – развитие количественного понимания качества программных продуктов проекта и достижение определенных показателей качества.

Ключевые процессы: Уровень 5

Вопросы, решаемые как организацией, так и отдельными проектами при реализации непрерывного и измеряемого усовершенствования производственного процесса:

1. Предотвращение дефектов
2. Управление технологическими изменениями
3. Управление изменениями процесса

Метрики разработки ПО

Мера и Метрика

- *Мера* - количественный показатель степени, количества, или размеров некоторых атрибутов продукта или процесса.
  - Например, количество ошибок

- *Метрика* - количественная мера позволяющая оценить, в какой степени система, компоненты или процесс обладают заданным атрибутом. “Предположение о значении данного атрибута.”
  - Например, количество обнаруженных ошибок на затраченный человеко-час

Зачем измерять ПО?

- Определение качества существующего продукта или процесса
- Прогнозирование качества продукта / процесса
- Улучшение качества продукта / процесса

Мотивация для метрик

- Оценка стоимости и графика будущих проектов
- Оценка производительности применения новых средств и методов
- Определение тенденций производительности с течением времени
- Улучшение качества программного обеспечения
- Прогноз будущих потребностей в персонале
- Предвидеть и сокращения будущих потребностей в техническом обслуживании

Классификация метрик

- Продукты
  - Результаты деятельности разработки программного обеспечения
  - Конечная продукция, документация по продуктам
- Процессы
  - Деятельность, связанная с производством программного обеспечения
- Ресурсы
  - База для процесса разработки программного обеспечения
  - Оборудование, знания, люди

Метрики сложности программного кода

Метрики сложности программ принято разделять на три основные группы:

1. метрики размера программ;
2. метрики сложности потока управления программ;
3. метрики сложности потока данных программ.

Размерно-ориентированные метрики

Размерно-ориентированные метрики

LOC-оценка (Lines Of Code) может включать в себя:

- общие трудозатраты (в человеко-месяцах, человеко-часах);
- объем программы (в тысячах строках исходного кода - LOC);
- стоимость разработки;
- объем документации;
- ошибки, обнаруженные в течение года эксплуатации;
- количество людей, работавших над изделием;
- срок разработки.

Физические и логические строки кода

- «**Физические**» строки кода – SLOC (используемые аббревиатуры **LOC, SLOC, KLOC, KSLOC, DSLOC**) – определяется как общее число строк исходного кода, включая комментарии и пустые строки.
- «**Логические**» строки кода – SLOC (используемые аббревиатуры **LSI, DSI, KDSI, где «SI»** - source instructions) – определяется как количество команд и зависит от используемого языка программирования.

Недостатки SLOC

- Неправильно сводить оценку работы человека к нескольким числовым параметрам. Менеджер может назначить наиболее талантливых программистов на сложнейший участок работы.
- Метрика не учитывает опыт сотрудников и их другие качества.
- Искажение: процесс измерения может быть искажён за счёт того, что сотрудники знают об измеряемых показателях и стремятся оптимизировать эти показатели, а не свою работу.
- Неточность: нет метрик, которые были бы одновременно и значимы и достаточно точны. Количество строк кода — это просто количество строк, этот показатель не даёт представления о сложности решаемой проблемы.

Метрики сложности

Метрики сложности

- **Объектно-ориентированные**: оценка сложности объектно-ориентированных проектов
- **Метрики Холстеда**: вычисляются на основании анализа числа строк и синтаксических элементов исходного кода программы
- **Цикломатическая сложность** : один из наиболее распространенных показателей оценки сложности программных проектов на основе графа управляющей логики
- **Метрика Чепина**: оценка информационной прочности

Объектно-ориентированные метрики

Метрика	Описание
Взвешенная насыщенность класса 1 (Weighted Methods Per Class (WMC)	Отражает относительную меру сложности класса на основе цикломатической сложности каждого его метода. Класс с более сложными методами и большим количеством методов считается более сложным. При вычислении метрики родительские классы не учитываются.
Взвешенная насыщенность класса 2 (Weighted Methods Per Class (WMC2))	Мера сложности класса, основанная на том, что класс с большим числом методов, является более сложным, и что метод с большим количеством параметров также является более сложным. При вычислении метрики родительские классы не учитываются.
Глубина дерева наследования (Depth of inheritance tree)	Длина самого длинного пути наследования, заканчивающегося на данном модуле. Чем глубже дерево наследования модуля, тем может оказаться сложнее предсказать его поведение. С другой стороны, увеличение глубины даёт больший потенциал повторного использования данным модулем поведения, определённого для классов-предков.
Количество детей (Number of children)	Число модулей, непосредственно наследующих данный модуль. Большие значения этой метрики указывают на широкие возможности повторного использования; при этом слишком большое значение может свидетельствовать о плохо выбранной абстракции.
Связность объектов (Coupling between objects)	Количество модулей, связанных с данным модулем в роли клиента или поставщика. Чрезмерная связность говорит о слабости модульной инкапсуляции и может препятствовать повторному использованию кода.
Отклик на класс (Response For Class)	Количество методов, которые могут вызываться экземплярами класса; вычисляется как сумма количества локальных методов, так и количества удаленных методов

Метрики Холстеда

Основу метрики Холстеда составляют четыре измеряемые характеристики программы:

- **NUOprtr** (Number of Unique Operators) — число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);
- **NUOprnd** (Number of Unique Operands) — число уникальных операндов программы (словарь операндов);
- **Noprtr** (Number of Operators) — общее число операторов в программе;
- **Noprnd** (Number of Operands) — общее число операндов в программе.

Оценки на основе Метрики Холстеда

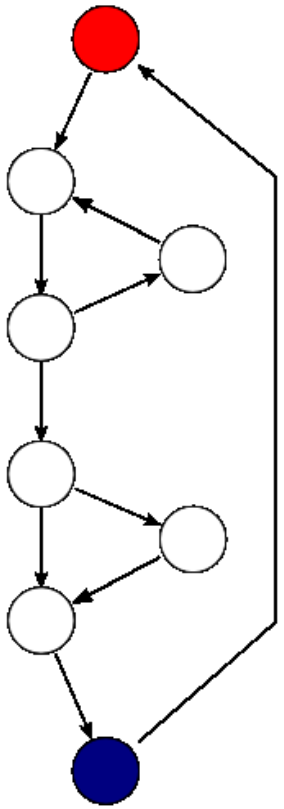
- **Словарь программы** (Halstead Program Vocabulary):
  - HPVoc = NUOprtr + NUOprnd;
- **Длина программы** (Halstead Program Length):
  - HPLen = Noprtr + Noprnd;
- **Объем программы** (Halstead Program Volume):
  - HPVol = HPLen log2 HPVoc;
- **Сложность программы** (Halstead Difficulty, HDiff):
  - HDiff = (NUOprtr/2) × (NOprnd / NUOprnd);
- На основе показателя HDiff предлагается оценивать **усилия программиста** при разработке при помощи показателя HEff (Halstead Effort):
  - HEff = HDiff × HPVol.

Цикломатическая сложность

- Цикломатическая сложность части программного кода — счётное число линейно независимых маршрутов через программный код.
- Если исходный код не содержит никаких точек решений, таких, как указания **IF** или циклы **FOR**, то сложность равна единице, поскольку, есть только единственный маршрут через код.

ЦС: Граф потока управления программы

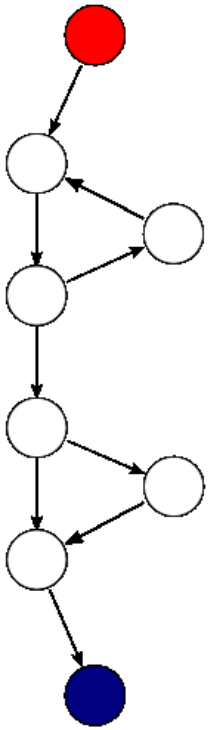
- Графом потока управления программы называется ориентированный граф, в узлах которого находятся неделимые группы команд, а ребрами соединяются такие блоки, которые могут быть выполнены сразу друг за другом.



(2) Цикломатическая сложность

- Показатель цикломатической сложности позволяет не только **произвести оценку** трудоемкости реализации отдельных элементов программного проекта и скорректировать общие показатели оценки длительности и стоимости проекта, но и **оценить связанные риски** и принять необходимые управленческие решения.
- **C = e – n + 2p**, где e – число ребер, n – число узлов, а p – число компонентов связности на графе управляющей логики.

Пример цикломатической сложности



Программа начинает выполняться с красного узла,
затем идут циклы (после красного узла идут две группы по три узла).
Выход из цикла осуществляется через условный оператор (нижняя группа узлов)
и конечный выход из программы в синем узле. Для этого графа E = 9, N = 8 и P = 1,
цикломатическая сложность программы равна 3.

Метрики Чепина

Все множество переменных, составляющих список ввода-вывода, разбивается на четыре функциональные группы.

- **Множество «P»** – вводимые переменные для расчетов и для обеспечения вывода. Примером может служить используемая в программах лексического анализатора переменная, содержащая строку исходного текста программы, то есть сама переменная не модифицируется, а только содержит исходную информацию.
- **Множество «M»** – модифицируемые или создаваемые внутри программы переменные.
- **Множество «C»** – переменные, участвующие в управлении работой программного модуля (управляющие переменные).
- **Множество «T»** – не используемые в программе (“паразитные”) переменные.

Поскольку каждая переменная может выполнять одновременно несколько функций, необходимо учитывать ее в каждой соответствующей функциональной группе.

Вычисление метрики Чепина

В общем виде

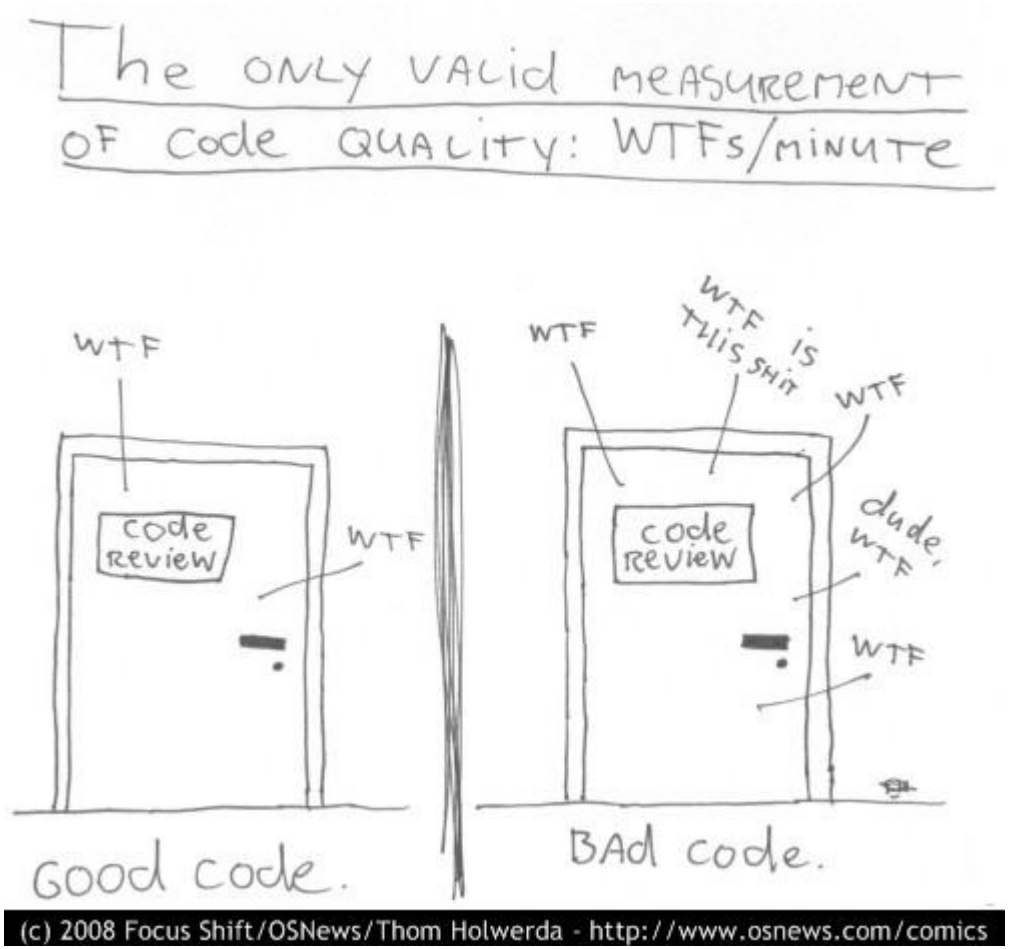
- Q = a1P + a2M + a3C + a4T,

где a1, a2, a3, a4 – весовые коэффициенты.

Автор предлагает следующие значения коэффициентов:

- **Q = P + 2M + 3C + 0.5T**

Единственно-правильная метрика качества программного кода



Резюме

Резюме

Резюме