Введение в программную инженерию

ПОНЯТИЕ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Чем программирование отличается от программой инженерии?

- Программирование является некоторой абстрактной деятельностью и может происходить во многих различных контекстах.
- Промышленное программирование, как правило, происходит в команде, и совершенно точно – для заказчика, который платит за работу деньги.
- Необходимо точно понимать, что нужно заказчику, выполнить работу в определенные сроки и результат должен быть нужного качества –того, которое удовлетворит заказчика и за которое он заплатит.
- Чтобы удовлетворить этим дополнительным требованиям, программирование «обрастает» различными дополнительными видами деятельности: разработкой требований, планированием, тестированием, конфигурационным управлением, проектным менеджментом, созданием различной документации (проектной, пользовательской и пр.).

- Разработка программного кода предваряется анализом и проектированием (первое означает создание функциональной модели будущей системы без учета реализации, для осознания программистами требований и ожиданий заказчика; второе означает предварительный макет, эскиз, план системы на бумаге).
- Требуются специальные усилия по организации процесса разработки.
- Разработку системы необходимо выполнять с учетом удобств u ее дальнейшего сопровождения, повторного использования и интеграции с другими системами.
- Это значит, что система разбивается на компоненты, удобные в разработке, годные для повторного использования и интеграции
- Для этих компонент тщательно прорабатываются интерфейсы.
- Сама же система документируется на многих уровнях, создаются правила оформления программного кода – то есть оставляются многочисленные семантические следы, помогающие создать и сохранить, поддерживать единую, стройную архитектуру, единообразный стиль, порядок…
- Все эти и другие дополнительные виды деятельности, выполняемые в процессе промышленного программирования и необходимые для успешного выполнения заказов и будем называть ***программной инженерией*** (software engineering)

Предпосылки появления программной инженерии

- ***Программная инженерия*** (или технология промышленного программирования) как некоторое направление возникло и формировалось под давлением роста стоимости создаваемого программного обеспечения. Главная цель этой области знаний – сокращение стоимости и сроков разработки программ.
- ***Программная инженерия*** прошла несколько этапов развития, в процессе которых были сформулированы фундаментальные принципы и методы разработки программных продуктов. Основной принцип программной инженерии состоит в том, что программы создаются в результате выполнения нескольких взаимосвязанных этапов (анализ требований, проектирование, разработка, внедрение, сопровождение), составляющих жизненный цикл программного продукта. Фундаментальными методами проектирования и разработки являются модульное, структурное и объектно-ориентированное проектирование и программирование.

Повторное использование кода (модульное программирование)

- ***Проблема.***

  На первых этапах становления программной инженерии было отмечено, что высокая стоимость программ связана с разработкой одинаковых (или похожих) фрагментов кода в различных программах. Вызвано это было тем, что в различных программах как части этих программ решались одинаковые (или похожие) задачи: решение нелинейных уравнений, расчет заработной платы, … Использование при создании новых программ ранее написанных фрагментов сулило существенное снижение сроков и стоимости разработки.

Рост сложности программ (структурное программирование)

- ***Проблема.***

  Следующий этап возрастания стоимости ПО был связан с переходом от разработки относительно простых программ к разработке сложных программных комплексов. Следует отметить, что этот переход был вызван появлением вычислительной техники третьего поколения (интегральные схемы). С переходом на использование интегральных схем производительность компьютеров возросла на порядки, что и создало предпосылки для решения сложных задач. К числу таких сложных задач относятся: системы управления космическими объектами, управления оборонным комплексом, автоматизации крупного финансового учреждения и т.д.

Сложность таких комплексов оценивалась следующими показателями:

- Большой объем кода (миллионы строк)
- Большое количество связей между элементами кода
- Большое количество разработчиков (сотни человек)
- Большое количество пользователей (сотни и тысячи)
- Длительное время использования

Для таких сложных программ оказалось, что основная часть их стоимости приходится не на создание программ, а на их внедрение и эксплуатацию. По аналогии с промышленной технологией стали говорить о жизненном цикле программного продукта, как о последовательности определенных этапов: этапа проектирования, разработки, тестирования, внедрения и сопровождения.

Модификация программ (ООП)

- **Проблема.**

Следующая проблема роста стоимости программ была вызвана тем, что изменение требований к программе стали возникать не только на стадии сопровождения,

но и на стадии проектирования – проблема заказчика, который не знает, что он хочет.

Создание программного продукта превратилось в его перманентное перепроектирова -ние. Возник вопрос, как проектировать и писать программы, чтобы обеспечить возможность внесений изменений в программу, не меняя ранее написанного кода.

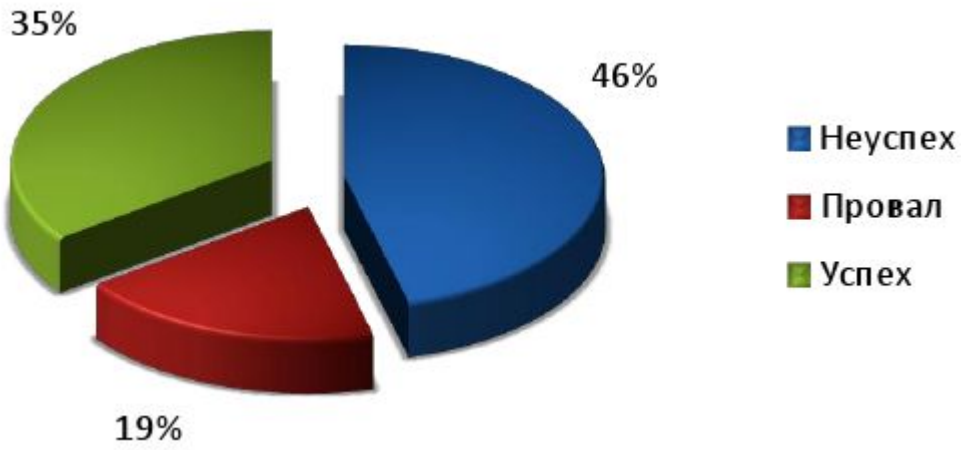
Продолжение кризиса программирования

- Несмотря на то, что программная инженерия достигла определенных успехов, перманентный кризис программирования продолжается. Связано это с тем, что рубеж 80–90-х годов отмечается как начало информационно- технологической революции, вызванной взрывным ростом использования информационных средств: персональный компьютер, локальные и глобальные вычислительные сети, мобильная связь, электронная почту, Internet и т.д. Цена успеха – кризис программирования принимает хронические формы.

  Standish Group, проанализировав работу сотен американских корпораций и итоги выполнения нескольких десятков тысяч проектов, связанных с разработкой ПО, в своем докладе с красноречивым названием «Хаос» пришла к следующим выводам :

  – Только 35 % проектов завершились в срок, не превысили запланированный бюджет и реализовали все требуемые функции и возможности.
  – 46 % проектов завершились с опозданием, расходы превысили запланированный бюджет, требуемые функции не были реализованы в полном объеме. Среднее превышение сроков составило 120%, среднее превышение затрат 100%, обычно исключалось значительное число функций.

- 19 % проектов полностью провалились и были аннулированы до завершения.



Результаты анализа успешности программных проектов за 2006 год

Основные понятия

- *Программирование* — процесс отображения определенного множества целей на множество машинных команд и данных, интерпретация которых на компьютере или вычислительном комплексе обеспечивает достижение поставленных целей.
- Это отображение может быть очень простым, а может быть многоступен - чатым и очень сложным, когда сначала цели отображаются на требования к системе, требования — на высокоуровневую архитектуру и спецификации компонентов, спецификации — на дизайн компонентов, дизайн — на исходный код.
- *Профессиональное программирование* (синоним производство программ) — деятельность, направленная на получение доходов при помощи программирования.
- *Профессиональный программист* — человек, который занимается профессиональным программированием.

- *Программный продукт* — совокупность программ и сопроводительной документации по их установке, настройке, использованию и доработке.
- Согласно стандарту жизненный цикл программы, программной системы, программного продукта включает в себя разработку, развертывание, поддержку и сопровождение.

Жизненный цикл программного продукта

- *Процесс разработки ПО* — совокупность процессов, обеспечивающих создание и развитие программного обеспечения.
- *Модель процесса разработки ПО* — формализованное представление процесса разработки ПО. Часто при описании процессов вместо слова модель употребляется термин методология.
- На сегодняшний день нет единого определения понятия «программная инженерия». Ниже приведено несколько таких
- Определения, данные крупными специалистами в этой области, или зафиксированные в документах ведущих организаций:
  - установление и использование обоснованных инженерных принципов (методов) для экономного получения ПО, которое надежно и работает на реальных машинах. [Bauer 1972];
  - та форма инженерии, которая применяет принципы информатики (computer science) и математики для рентабельного решения проблем ПО. [CMU/SEI-90-TR-003]
  - применение систематического, дисциплинированного, измеряемого подхода к разработке, использованию и сопровождению ПО [IEEE 1990].
  - дисциплина, целью которой является создание качественного ПО, которое завершается вовремя, не превышает выделенных бюджетных средств и удовлетворяет выдвигаемым требованиям [Schach, 99].

Хронология развития программной инженерии

- 1958 г. - всемирно известный статистик Джон Тьюкей (John Tukey) впервые ввел термин software – программное обеспечение.
- 1968 г. - Термин software engineering (программная инженерия) впервые появился в названии конференции НАТО, состоявшейся в Германии посвященной так называемому кризису программного обеспечения.
- 1970 г. - У.У. Ройс (W.W. Royce) произвел идентификацию нескольких стадий в типичном цикле и было высказано предположение, что контроль выполнения стадий приведет к повышению качества ПО и сокращению стоимости разработки. Была предложена концепция жизненного цикла (SLC – Software Lifetime Cycle).
- 1990 - 1995 г. - велась работа над международным стандартом, который должен был дать единое представление о процессах разработки программного обеспечения. В результате был выпущен стандарт ISO/IEC 12207 —Software Lifecycle Processes.
  - Соответствующий национальный стандарт России – ГОСТ Р ИСО/МЭК 12207-99 [ГОСТ 12207, 1999] содержит полный аутентичный перевод текста международного стандарта ISO/IEC 12207-95

2004 г - ACM/IEEE-CS сформулировали два ключевых описания того, что сегодня мы и называем основами программной инженерии – Software Engineering:

1. Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE 2004 Version - Руководство к Своду Знаний по Программной Инженерии, в дальнейшем просто ―SWEBOK;

   Одной из важнейших целей SWEBOK является именно определение тех аспектов деятельности, которые составляют суть профессии инженера-программиста.

2. Software Engineering 2004.– Учебный План для Преподавания Программной Инженерии в ВУЗах [SE, 2004].

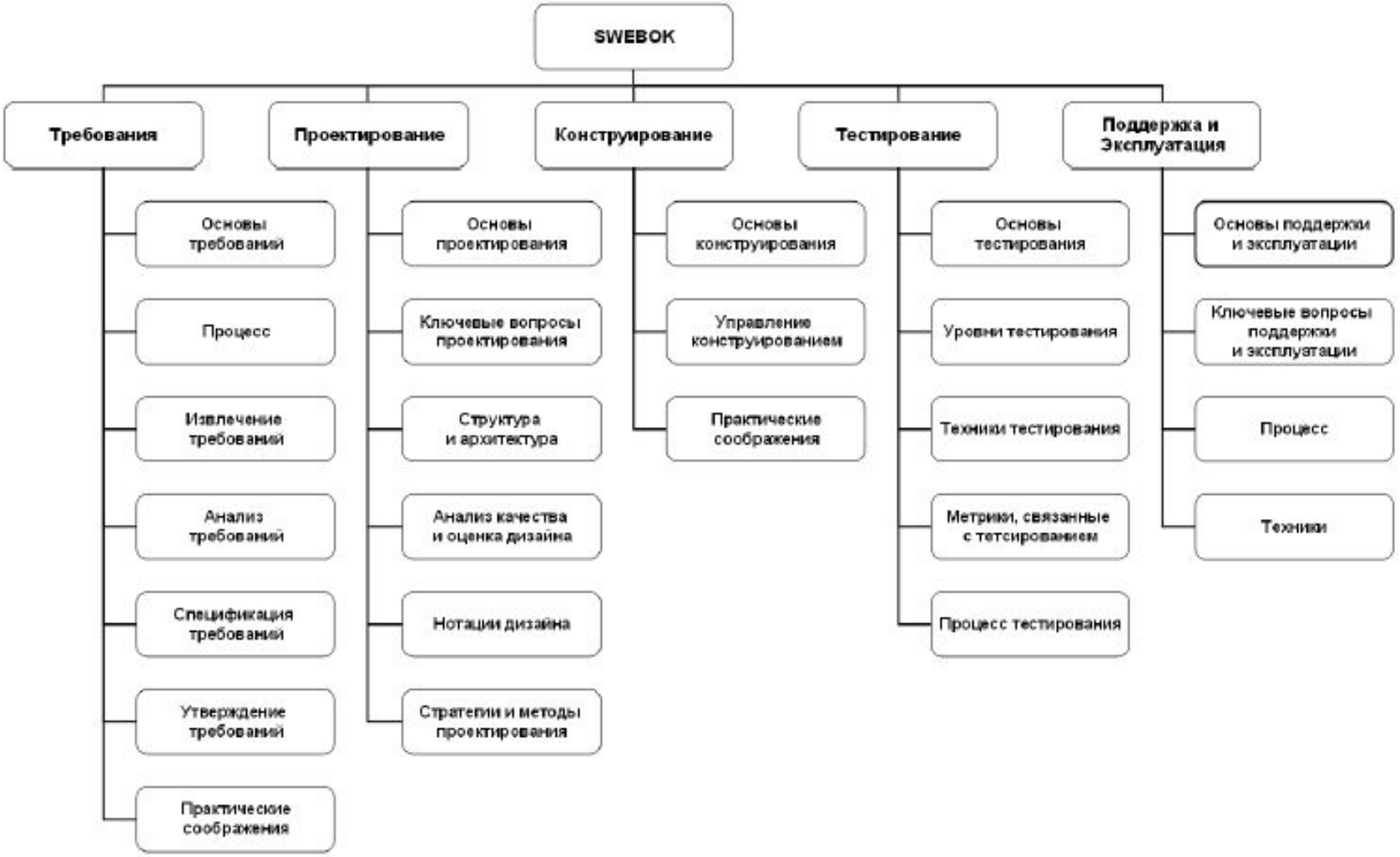
Структура и содержание SWEBOK

Описание областей знаний в SWEBOK построено по иерархическому принципу, как результат структурной декомпозиции.

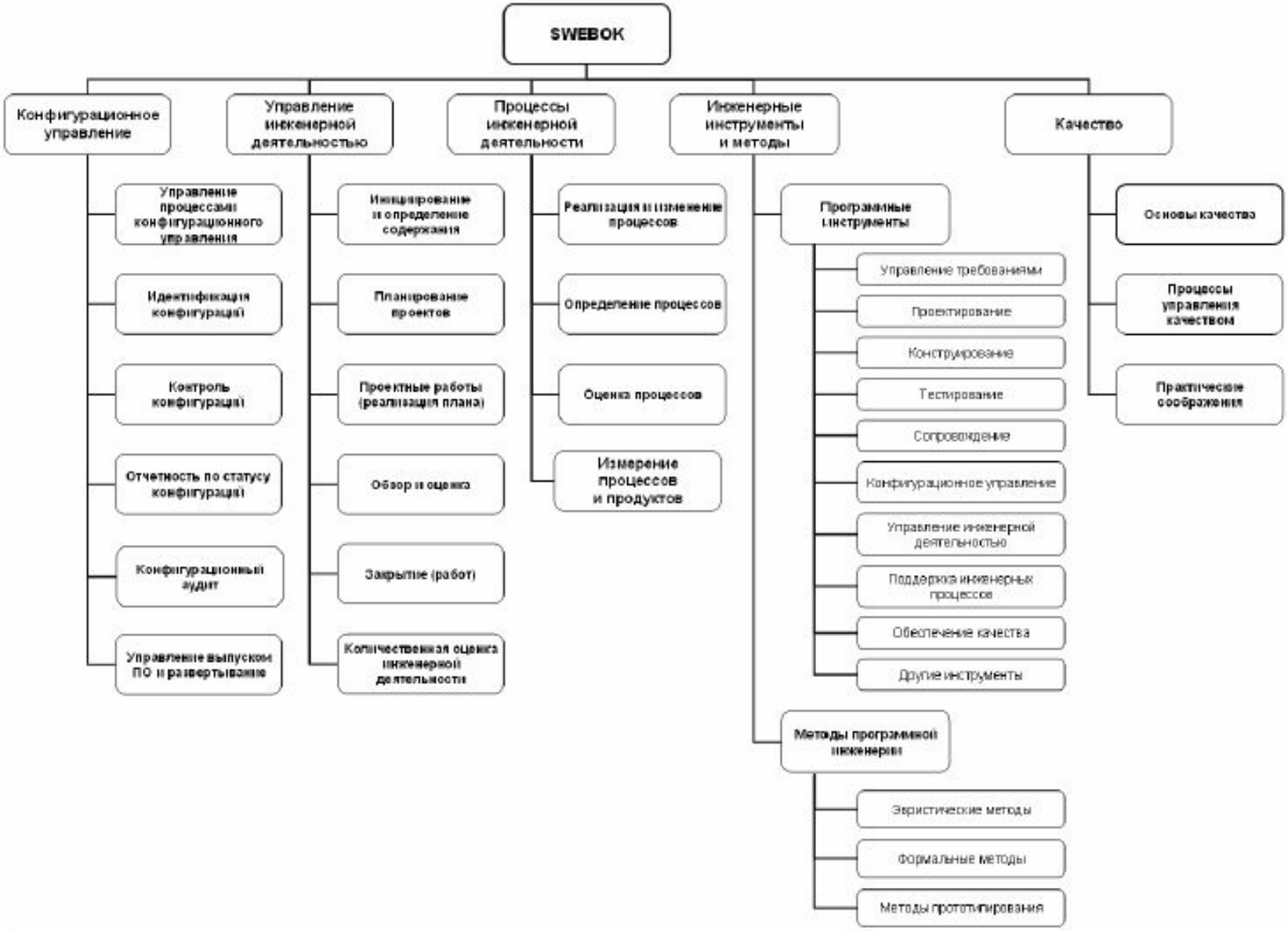
Согласно SWEBOK 2004, программная инженерия включает в себя 10 основных и 7 дополнительных областей знаний, на которых базируются процессы разработки ПО. К основным областям знаний относятся следующие области:

- Software requirements — программные требования.
- Software design — дизайн (архитектура).
- Software construction — конструирование программного обеспечения.
- Software testing — тестирование.
- Software maintenance — эксплуатация (поддержка) программного обеспечения.
- Software configuration management — конфигурационное управление.
- Software engineering management — управление в программной инженерии.
- Software engineering process — процессы программной инженерии.
- Software engineering tools and methods — инструменты и методы.
- Software quality — качество программного обеспечения.

- **Первые пять областей знаний**



- **Вторые пять областей знаний**

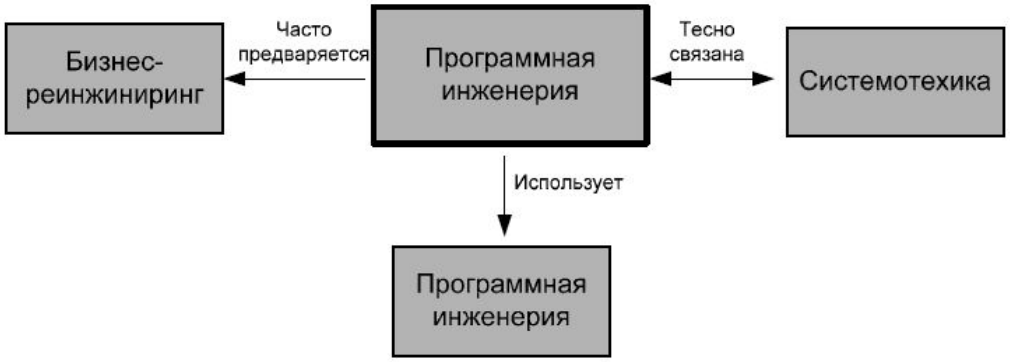


Связанные дисциплины

- Дополнительные области знаний включают в себя:
- Computer engineering — разработка компьютеров.
- Computer science — информатика.
- Management — общий менеджмент.
- Mathematics — математика.
- Project management — управление проектами.
- Quality management — управление качеством.
- Systems engineering — системное проектирование.


SWEBOK
Смежные дисциплины
Компьютерная инженерия
Информатика
Управление
Математика
Управление проектами
Управление качеством
Эргономика ПО
Системное проектирование

- Программная инженерия использует достижения информатики, тесно связана с системотехникой, часто предваряется бизнес-реинжинирингом.



- ***Информатика*** (computer science) – это свод теоретических наук, основанных на математике и посвященных формальным основам вычислимости. Сюда относятся математическую логику, теорию грамматик, методы построения компиляторов, математические формальные методы, используемые в верификации и модельном тестировании и т.д. Трудно строго отделить программную инженерию от информатики, но в целом направленность этих дисциплин различна. Программная инженерия нацелена на решение проблем производства, информатика – на разработку формальных, математизирован - ных подходов к программированию.

- ***Системотехника*** (system engineering) объединяет различные инженерные дисциплины по разработке всевозможных искусственных систем – энергоустановок, телекоммуникационных систем, встроенных систем реального времени и т.д. Очень часто ПО оказывается частью таких систем, выполняя задачу управления соответствующего оборудования. Такие системы называются *программно-аппаратными*, и участвуя в их создании, програм - мисты вынуждены глубоко разбираться в особенностях соответствующей аппаратуры.
- ***Бизнес-реинжиниринг*** (business reengineering) – в широком смысле обозначает модернизацию бизнеса в определенной компании, внедрение новых практик, поддерживаемых соответствующими, новыми информаци - онными системами. При этом акцент может быть как на внутреннем переустройстве компании так и на разработке нового клиентского сервиса (как правило, эти вопросы взаимосвязаны). Бизнес-реинжиниринг часто предва- ряет разработку и внедрение информационных систем на предприятии, так как требуется сначала навести определенный порядок в делопроизводстве, а лишь потом закрепить его информационной системой.

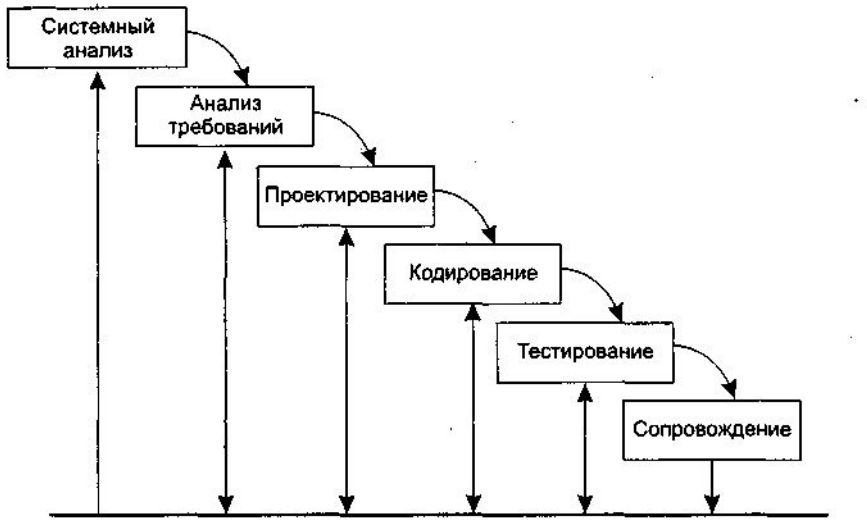
ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ПРОДУКТА

- Методологическую основу любой инженерии составляет понятие жизненного цикла (ЖЦ) изделия (продукта) как совокупности всех действий, которые надо выполнить на протяжении всей «жизни» изделия.
- Смысл жизненного цикла состоит во взаимосвязанности всех этих действий.
- Жизненный цикл промышленного изделия определяется как последовательность этапов (фаз, стадий), состоящих из технологических процессов, действий и операци
- Организация промышленного производства с позиции жизненного цикла позволяет рассматривать все его этапы во взаимосвязи, что ведет к сокращению сроков, стоимости и трудозатрат.
- Разделение понятий ЖЦ ПО и модели ЖЦ ПО.
  - ЖЦ ПО - полная совокупность всех процессов и действий по созданию и применению ПО;
  - модель ЖЦ – конкретный вариант организации ЖЦ, обоснованно (разумно) выбранный для каждого конкретного случая.

- В общем случае, жизненный цикл определяется моделью и описывается в форме методологии (метода). Модель или парадигма жизненного цикла определяет концептуальный взгляд на организацию жизненного цикла и, часто, основные фазы жизненного цикла и принципы перехода между ними.
- Методология (метод) задает комплекс работ, их детальное содержание и ролевую ответственность специалистов на всех этапах выбранной модели жизненного цикла, обычно определяет и саму модель, а также рекомендует практики (best practices), позволяющие максимально эффективно воспользоваться соответствующей методологией и ее моделью.

Классический жизненный цикл

- Классический жизненный цикл разработки ПО - старейшая парадигма процесса разработки ПО (автор Уинстон Ройс, 1970)
- Классический жизненный цикл называют каскадной или водопадной моделью;
- Разработка рассматривается как последовательность этапов, причем переход на следующий, иерархически нижний этап происходит только после полного завершения работ на текущем этапе



Содержание основных этапов.

- ***Системный анализ*** - задает роль каждого элемента в компьютерной системе, взаимодействие элементов друг с другом.
- На этом этапе начинается решение задачи планирования проекта ПО. Определяются:
  - объем проектных работ;
  - риск;
  - необходимые трудозатраты;
  - формируются рабочие задачи;
  - план-график работ.
- ***Анализ требований*** - относится к программному элементу — программному обеспечению.
- Уточняются и детализируются
  - Функции;
  - Характеристики;
  - Интерфейс.
- Все определения документируются в *спецификации анализа.* Здесь же завершается решение задачи планирования проекта.

- ***Проектирование*** состоит в создании представлений:
  - архитектуры ПО;
  - модульной структуры ПО;
  - алгоритмической структуры ПО;
  - структуры данных;
  - входного и выходного интерфейса (входных и выходных форм данных).
- Исходные данные для проектирования содержатся в *спецификации анализа.*
- При решении задач проектирования основное внимание уделяется качеству будущего программного продукта.
- ***Кодирование*** состоит в переводе результатов проектирования в текст на языке программирования.
- ***Тестирование*** — выполнение программы для выявления дефектов в функциях, логике и форме реализации программного продукта.
- ***Сопровождение*** — это внесение изменений в эксплуатируемое ПО.
- Цели изменений:
  - исправление ошибок;
  - адаптация к изменениям внешней для ПО среды;
  - усовершенствование ПО по требованиям заказчика.

- ***Сопровождение ПО*** состоит в повторном применении каждого из предшествующих шагов (этапов) жизненного цикла к существующей программе но не в разработке новой программы.

- ***Достоинства классического жизненного цикла:***
  - дает план и временной график по всем этапам проекта;
  - упорядочивает ход конструирования

- ***Недостатки классического жизненного цикла:***
  - реальные проекты часто требуют отклонения от стандартной последовательности шагов;
  - цикл основан на точной формулировке исходных требований к ПО (реально в начале проекта требования заказчика определены лишь частично);
  - результаты проекта доступны заказчику только в конце работы.

Макетирование

Проблемы

- *Заказчик* - не может сформулировать подробные требования по вводу, обработке или выводу данных для будущего программного продукта.
- *Разработчик* может сомневаться:
  - в приспосабливаемости продукта под операционную систему;
  - форме диалога с пользователем;
  - в эффективности реализуемого алгоритма.

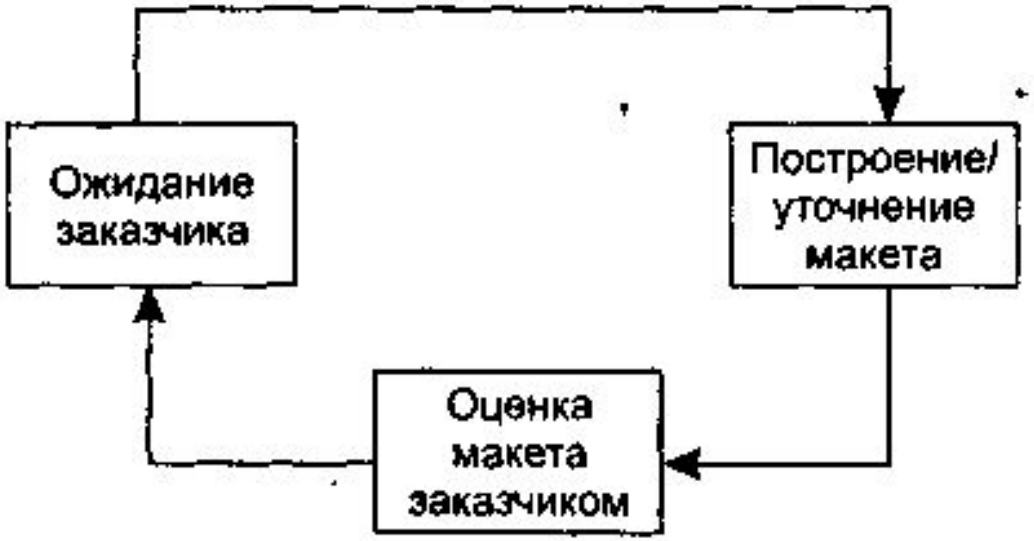
В этих случаях целесообразно использовать макетирование.

- Основная цель макетирования — снять неопределенности в требованиях заказчика.
- **Макетирование** (прототипирование) — это процесс создания модели требуемого программного продукта.
- Модель может принимать одну из трех форм:

1) бумажный макет или макет на основе ПК (изображает или рисует человеко-машинный диалог);
2) работающий макет (выполняет некоторую часть требуемых функций);
3) существующая программа (характеристики которой затем должны быть улучшены).

- Макетирование основывается на многократном повторении итераций, в которых участвуют заказчик и разработчик.

Макетирование



Последовательность действий при макетировании

- *Сбор и уточнение требований к создаваемому ПО.*
  Разработчик и заказчик встречаются и определяют все цели ПО, устанавлива -ют, какие требования известны, а какие предстоит доопределить.
- *Быстрое проектирование.*
  Внимание сосредоточивается на тех характеристиках ПО, которые должны быть видимы пользователю. Быстрое проектирование приводит к построе- нию макета.
- *Макетирование.*
- *Оценивается макета заказчиком* - используется для уточнения требований к ПО.
- *Уточнение макета.*
- Итерации повторяются до тех пор, пока макет не выявит все требования заказчика и, тем самым, не даст возможность разработчику понять, что должно быть сделано.

***Достоинство макетирования**:* обеспечивает определение полных требований к ПО.

Недостатки макетирования:

- заказчик может принять макет за продукт;
- разработчик может принять макет за продукт.

- Для быстрого получения работающего макета разработчик часто идет на определенные компромиссы:
  - используются не самые подходящие языки программирования или операционная система;
  - для простой демонстрации возможностей может применяться неэффективный алгоритм.

Стратегии конструирования ПО

Существуют 3 стратегии конструирования ПО:

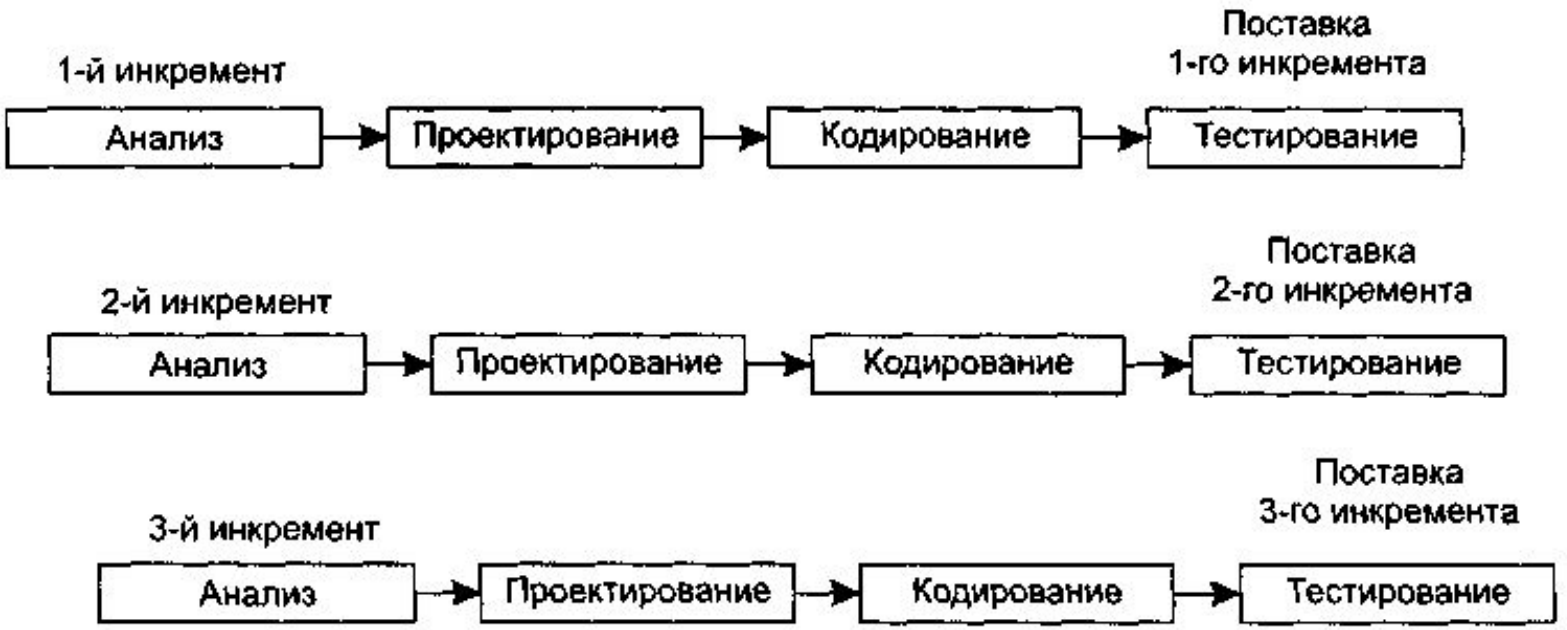
- *однократный проход* (водопадная стратегия) — линейная последовательность этапов конструирования;
- *инкрементная стратегия.* В начале процесса определяются все пользовательские и системные требования, оставшаяся часть конструирования выполняется в виде последовательности версий. Первая версия реализует часть запланированных возможностей, следующая версия реализует дополнительные возможности и т. д., пока не будет получена полная система;
- *эволюционная стратегия.* Система также строится в виде последовательности версий, но в начале процесса определены не все требования. Требования уточняются в результате разработки версий.

Характеристики стратегий конструирования ПО в соответствии с требованиями стандарта IEEE/EIA 12207.2

Стратегия конструирования	В начале процесса определены все требования?	Множество циклов конструирования?	Промежуточное ПО распространяется?
Однократный проход	Да	Нет	Нет
Инкрементная (запланированное улучшение продукта)	Да	Да	Может быть
Эволюционная	Нет	Да	Да

Инкрементная модель

- *Инкрементная модель* является классическим примером инкрементной стратегии конструирования
- Она объединяет элементы последовательной водопадной модели с итерационной философией макетирования.



- Каждая линейная последовательность здесь вырабатывает поставляемый инкремент ПО.

Пример.

- ПО для обработки слов в 1-м инкременте реализует функции базовой обработки файлов, функции редактирования и документирования;
- во 2-м инкременте — более сложные возможности редактирования и документирования;
- в 3-м инкременте — проверку орфографии и грамматики;
- в 4-м инкременте — возможности компоновки страницы.

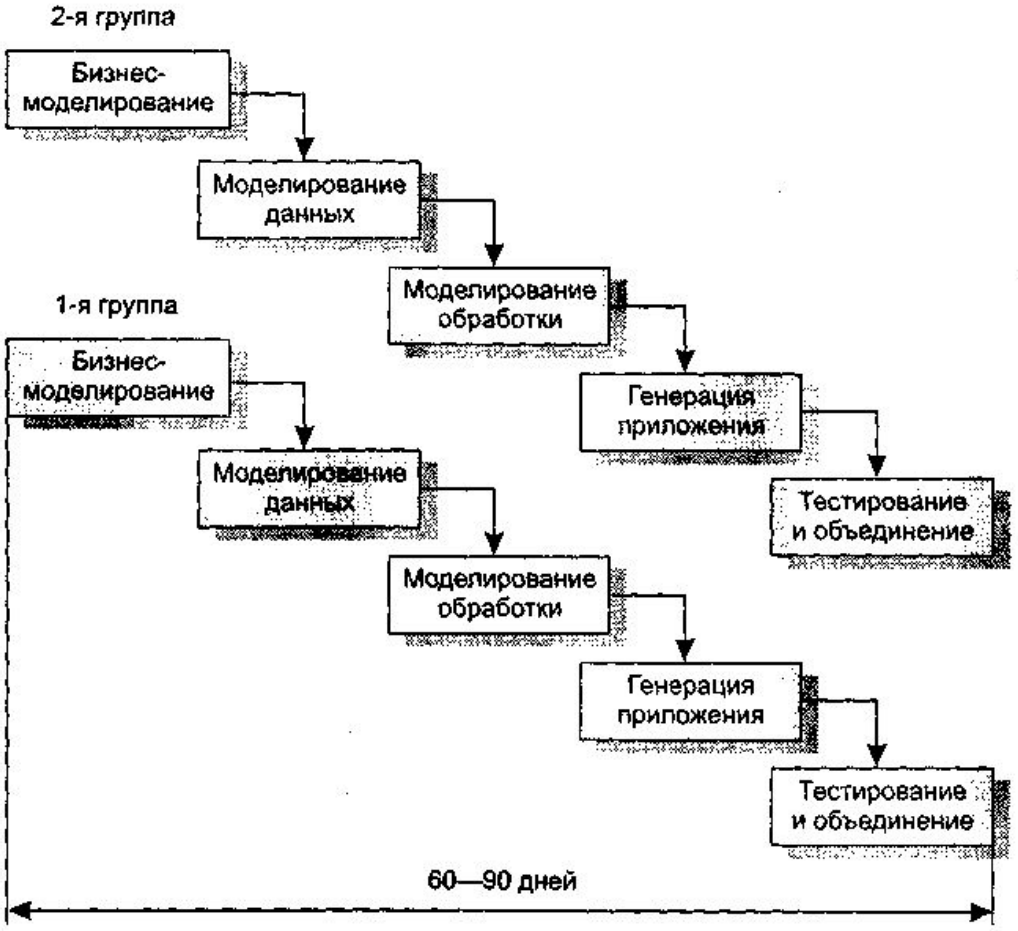
- Первый инкремент приводит к получению базового продукта, реализующего базовые требования (правда, многие вспомогательные требования остаются нереализованными).
- План следующего инкремента предусматривает модификацию базового продукта, обеспечивающую дополнительные характеристики и функциональность.

По своей природе инкрементный процесс итеративен, но, в отличие от макетирования, инкрементная модель обеспечивает на каждом инкременте работающий продукт.

Современная реализация инкрементного подхода — экстремальное программирование XP. (Кент Бек, 1999)

Быстрая разработка приложений

- Модель быстрой разработки приложений (Rapid Application Development) — второй пример применения инкрементной стратегии конструирования



- RAD-модель обеспечивает экстремально короткий цикл разработки.
- RAD — высокоскоростная адаптация линейной последовательной модели, в которой быстрая разработка достигается за счет использования компонентно-ориентированного конструирования.
- Если требования полностью определены, а проектная область ограничена, RAD-процесс позволяет группе создать полностью функциональную систему за очень короткое время (60-90 дней).
- RAD-подход ориентирован на разработку информационных систем и выделяет следующие этапы:
  - **бизнес-моделирование.** Моделируется информационный поток между бизнес-функциями. Ищется ответ на следующие вопросы: Какая информация руководит бизнес-процессом? Какая генерируется информация? Кто генерирует ее? Где информация применяется? Кто обрабатывает ее?
  - **моделирование данных**. Информационный поток, определенный на этапе бизнес-моделирования, отображается в набор объектов данных, которые требуются для поддержки бизнеса. Идентифицируются характеристики (свойства, атрибуты) каждого объекта, определяются отношения между объектами;

- **моделирование обработки.** Определяются преобразования объектов данных, обеспечивающие реализацию бизнес-функций. Создаются описания обработки для добавления, модификации, удаления или нахождения (исправления) объектов данных;
- **генерация приложения.** Предполагается использование методов, ориентированных на языки программирования 4-го поколения. Вместо создания ПО с помощью языков программирования 3-го поколения, RAD-процесс работает с повторно используемыми программными компонентами или создает повторно используемые компоненты. Для обеспечения конструирования используются утилиты автоматизации;
- **тестирование и объединение.** Поскольку применяются повторно используемые компоненты, многие программные элементы уже протестированы. Это уменьшает время тестирования (хотя все новые элементы должны быть протестированы).

- Применение RAD возможно в том случае, когда каждая главная функция может быть завершена за 3 месяца. Каждая главная функция адресуется отдельной группе разработчиков, а затем интегрируется в целую систему.

- Применение RAD имеет свои недостатки, и ограничения.

1. Для больших проектов в RAD требуются существенные людские ресурсы (необходимо создать достаточное количество групп).
2. RAD применима только для таких приложений, которые могут декомпозироваться на отдельные модули и в которых производительность не является критической величиной.
3. RAD не применима в условиях высоких технических рисков (то есть при использовании новой технологии).

Спиральная модель

- Спиральная модель — классический пример применения эволюционной стратегии конструирования.
- Спиральная модель (автор Барри Боэм, 1988) базируется на лучших свойствах классического жизненного цикла и макетирования, к которым добавляется новый элемент — анализ риска, отсутствующий в этих парадигмах

- Спиральная модель:

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

- Модель определяет четыре действия, представляемые четырьмя квадрантами спирали.

1. *Планирование* — определение целей, вариантов и ограничений.
2. *Анализ риска* — анализ вариантов и распознавание/выбор риска.
3. *Конструирование* — разработка продукта следующего уровня.
4. *Оценивание* — оценка заказчиком текущих результатов конструирования.

- Интегрирующий аспект спиральной модели очевиден при учете радиального измерения спирали. С каждой итерацией по спирали (продвижением от центра к периферии) строятся все более полные версии ПО.
- Первый виток спирали - определяются начальные цели, варианты и ограничения, распознается и анализируется риск.
- Если анализ риска показывает неопределенность требований, на помощь разработчику и заказчику приходит макетирование (используемое в квадранте конструирования).
- Следующая фаза планирования и анализа риска базируется на предложениях заказчика.
- В каждом цикле по спирали результаты анализа риска формируются в виде «продолжать, не продолжать».
- Если риск слишком велик, проект может быть остановлен.
- Движение по спирали продолжается, с каждым шагом продвигая разработчиков к более общей модели системы.

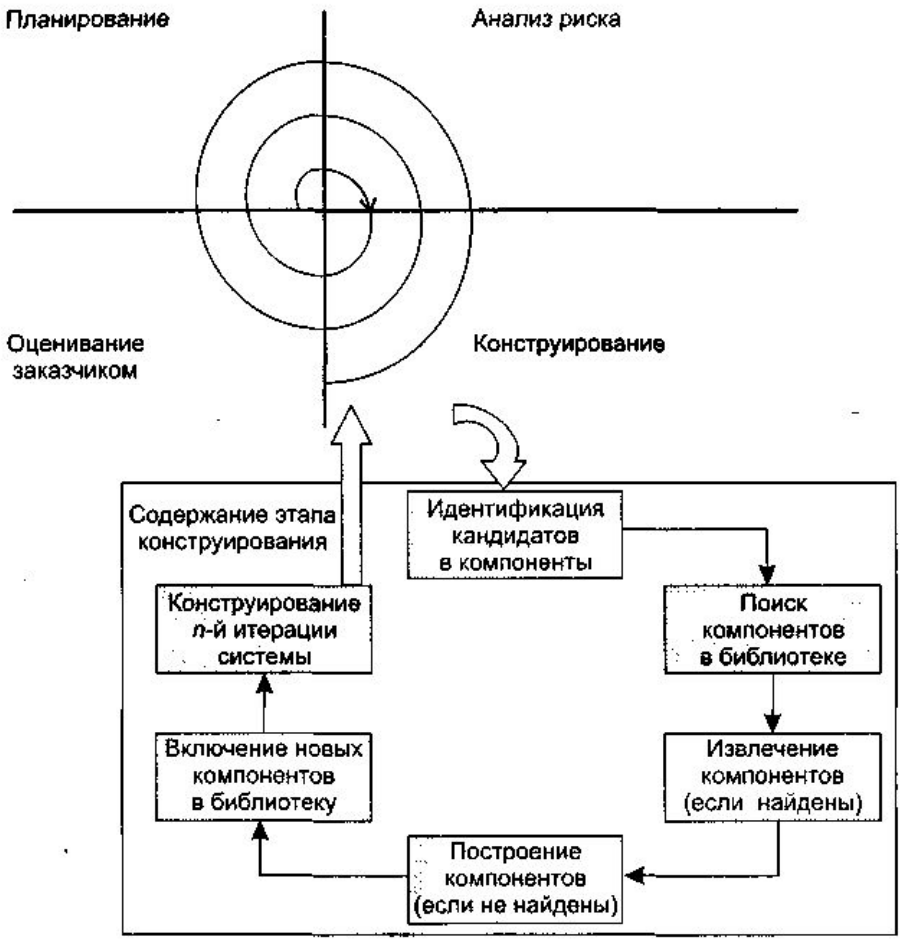
- *Достоинства спиральной модели:*

1) наиболее реально (в виде эволюции) отображает разработку программного обеспечения;
2) позволяет явно учитывать риск на каждом витке эволюции разработки;
3) включает шаг системного подхода в итерационную структуру разработки;
4) использует моделирование для уменьшения риска и совершенствования программного изделия.

- *Недостатки спиральной модели:*

1) новизна (отсутствует достаточная статистика эффективности модели);
2) повышенные требования к заказчику;
3) трудности контроля и управления временем разработки.

Компонентно-ориентированная модель



- Компонентно-ориентированная модель является развитием спиральной модели и тоже основывается на эволюционной стратегии конструирования.
- В модели конкретизируется содержание квадранта конструирования — оно отражает тот факт, что в современных условиях новая разработка должна основываться на повторном использовании существующих программных компонентов .
  - Программные компоненты, созданные в реализованных программных проектах, хранятся в библиотеке.
  - В новом программном проекте, исходя из требований заказчика, выявляются кандидаты в компоненты.
  - Далее проверяется наличие этих кандидатов в библиотеке.
  - Если кандидаты найдены, то компоненты извлекаются из библиотеки и используются повторно.
  - В противном случае создаются новые компоненты, они применяются в проекте и включаются в библиотеку.
- *Достоинства компонентно-ориентированной модели:*

1) уменьшает на 30% время разработки программного продукта;
2) уменьшает стоимость программной разработки до 70%;
3) увеличивает в полтора раза производительность разработки.

Тяжеловесные и облегченные процессы

- В XXI веке потребности общества в программном обеспечении информационных технологий достигли экстремальных значений.
- Традиционно для упорядочения и ускорения программных разработок предлагались строго упорядочивающие ***тяжеловесные*** (heavyweight) процессы.
- В этих процессах прогнозируется весь объем предстоящих работ, поэтому они называются прогнозирующими (predictive) процессами.
- Порядок, который должен выполнять при этом человек-разработчик, чрезвычайно строг и человеческие слабости в расчет не принимался, а объем необходимой документации был очень велик.
- В последние годы появилась группа новых, ***облегченных*** (lightweight) процессов, которые теперь называют ***подвижными*** (agile) процессами.
- Они привлекательны отсутствием бюрократизма, характерного для тяжеловесных (прогнозирующих) процессов и воплощают в жизнь разумный компромисс между слишком строгой дисциплиной и полным ее отсутствием.

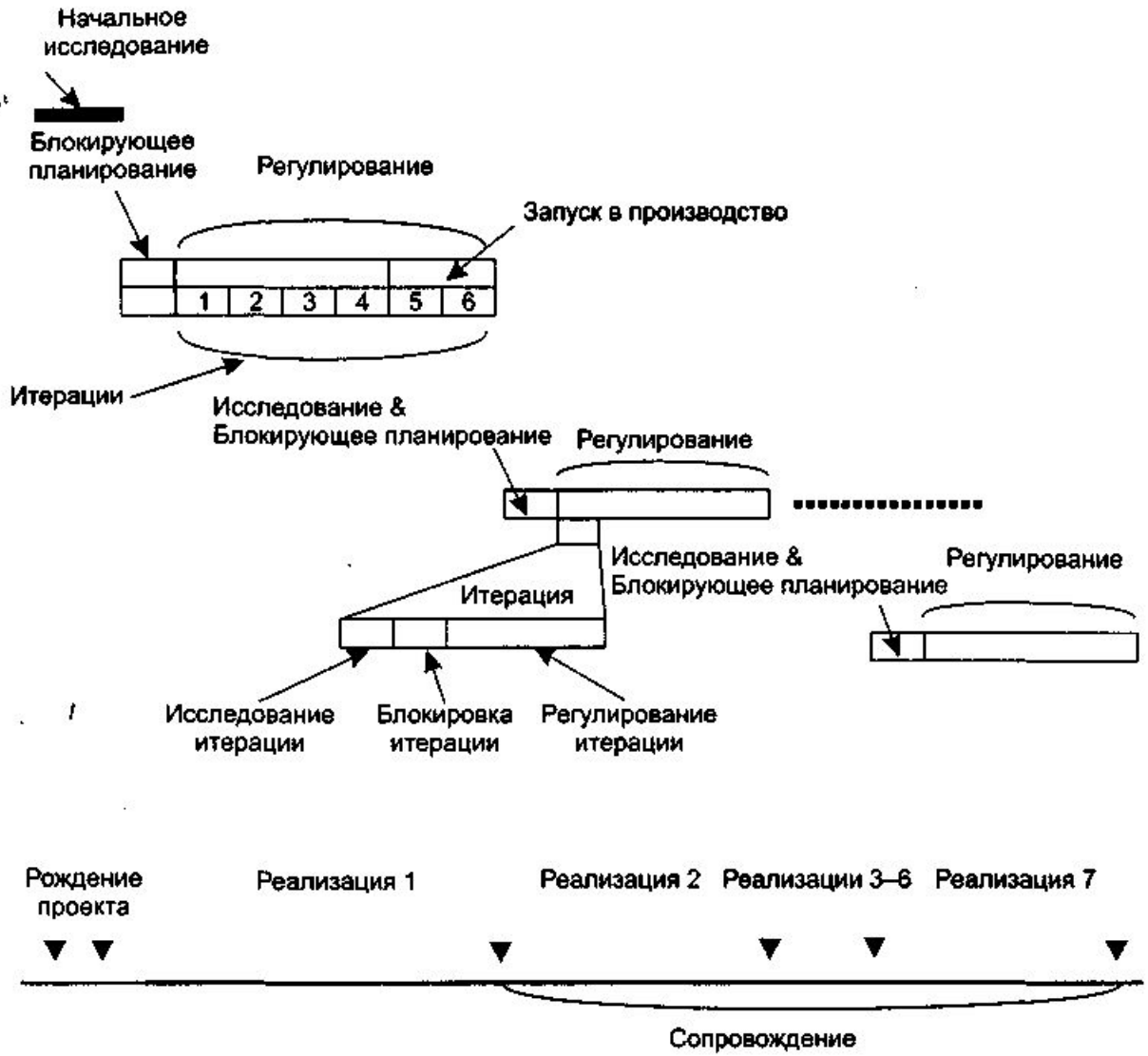
- Таким образом, в современной инфраструктуре программной инженерии существуют два семейства процессов разработки:
  - семейство прогнозирующих (тяжеловесных) процессов;
  - семейство адаптивных (подвижных, облегченных) процессов.
- У каждого семейства есть свои достоинства, недостатки и область применения:
  - адаптивный процесс используют при частых изменениях требований, малочисленной группе высококвалифицированных разработчиков и грамотном заказчике, который согласен участвовать в разработке;
  - прогнозирующий процесс применяют при фиксированных требованиях и многочисленной группе разработчиков разной квалификации.

XP-процесс

- *Экстремальное программирование* (eXtreme Programming, XP) — облегченный (подвижный) процесс (или методология), (Кент Бек, 1999).
- *XP-процесс ориентирован* на группы малого и среднего размера, строящие программное обеспечение в условиях неопределенных или быстро изменяющихся требований.
- *XP-группу образуют* до 10 сотрудников, которые размещаются в одном помещении.
- ***Основная идея XP*** — устранить высокую стоимость изменения, характерную для приложений с использованием объектов, паттернов и реляционных баз данных.
- *XP-процесс должен быть высокодинамичным процессом.* XP-группа имеет дело с изменениями требований на всем протяжении итерационного цикла разработки, причем цикл состоит из очень коротких итераций.
- ***Четырьмя базовыми действиями*** в XP-цикле являются: кодирование, тестирование, выслушивание заказчика и проектирование.
- *Динамизм обеспечивается с помощью четырех характеристик*: непрерывной связи с заказчиком (и в пределах группы), простоты (всегда выбирается минимальное решение), быстрой обратной связи (с помощью модульного и функционального тестирования), смелости в проведении профилактики возможных проблем.

- Большинство принципов, поддерживаемых в XP (минимальность, простота, эволюционный цикл разработки, малая длительность итерации, участие пользователя, оптимальные стандарты кодирования и т. д.), продиктованы здравым смыслом и применяются в любом упорядоченном процессе.
- В XP эти принципы достигают «экстремальных значений».
- Большинство принципов, поддерживаемых в XP (минимальность, простота, эволюционный цикл разработки, малая длительность итерации, участие пользователя, оптимальные стандарты кодирования и т. д.), продиктованы здравым смыслом и применяются в любом упорядоченном процессе.
- В XP эти принципы достигают «экстремальных значений».
- Рассмотрим структуру «идеального» XP-процесса.
- Основным структурным элементом процесса является XP-реализация, в которую многократно вкладывается базовый элемент — XP-итерация.
- В состав XP-реализации и XP-итерации входят три фазы —
  - исследование,
  - блокировка,
  - регулирование.

- **Структура «идеального» XP-процесса.**



- *Исследование (exploration)* — это поиск новых требований (историй, задач), которые должна выполнять система.
- *Блокировка (commitment)* — выбор для реализации конкретного подмножества из всех возможных требований (иными словами, планирование).
- *Регулирование (steering)* — проведение разработки, воплощение плана в жизнь.
- **XP рекомендует**:
  - первая реализация должна иметь длительность 2-6 месяцев
  - продолжительность остальных реализаций — около двух месяцев
  - каждая итерация длится приблизительно две недели,
  - численность группы разработчиков не превышает 10 человек.
- **Пример XP-процесса** для проекта с семью реализациями, осуществляемый за 15 месяцев
  - Процесс инициируется начальной исследовательской фазой.
  - Фаза исследования, с которой начинается любая реализация и итерация, имеет клапан «пропуска», на этой фазе принимается решение о целесообразности дальнейшего продолжения работы.
  - Предполагается:
    - длительность первой реализации составляет 3 месяца,
    - длительность второй — седьмой реализаций — 2 месяца.

- Вторая — седьмая реализации образуют период сопровождения, характеризующий природу XP-проекта.
- Каждая итерация длится две недели, за исключением тех, которые относят к поздней стадии реализации — «запуску в производство» (в это время темп итерации ускоряется).
- Наиболее трудна первая реализация — пройти за три месяца от обычного старта (скажем, отдельный сотрудник не зафиксировал никаких требований, не определены ограничения) к поставке заказчику системы промышленного качества очень сложно.

Модели качества процессов конструирования

- В условиях жесткой конкуренции, очень важно гарантировать высокое качество процесса конструирования ПО**.**
- Гарантию дает сертификат качества процесса, подтверждающий его соответствие принятым международным стандартам.
- Каждый стандарт фиксирует свою модель обеспечения качества.
- Наиболее авторитетными являются модели стандартов:
  - ISO 9001:2000;
  - ISO/ IEC 15504 ;
  - Модель зрелости процесса конструирования ПО (Capability Maturity Model — CMM) Института программной инженерии при американском университете Карнеги-Меллон.
- Модель стандарта ISO 9001:2000 ориентирована на процессы разработки из любых областей человеческой деятельности.
- Стандарт ISO/IEC 15504 специализируется на процессах программной разработки и отличается более высоким уровнем детализации.
- Базовым понятием модели CMM считается *зрелость* компании
  - *Незрелой* называют компанию, где процесс конструирования ПО и принимаемые решения зависят только от таланта конкретных разработчиков. Как следствие, здесь высока вероятность превышения бюджета или срыва сроков окончания проекта.

  - В зрелой компании работают ясные процедуры управления проектами и построения программных продуктов.
  - Оценки длительности и затрат разработки точны, основываются на накопленном опыте.
  - В компании имеются и действуют корпоративные стандарты на процессы взаимодействия с заказчиком, процессы анализа, проектирования, программирования, тестирования и внедрения программных продуктов.
- Модель CMM фиксирует критерии для оценки зрелости компании и предлагает рецепты для улучшения существующих в ней процессов
- Модель CMM ориентирована на построение системы постоянного улучшения процессов.
- В модели зафиксированы пять уровней зрелости и предусмотрен плавный, поэтапный подход к совершенствованию процессов — можно поэтапно получать подтверждения об улучшении процессов после каждого уровня зрелости.

Пять уровней зрелости модели CMM

- **Уровень 1** (**Начальный**) означает, что процесс в компании не формализован. Он не может строго планироваться и отслеживаться, его успех носит случайный характер. Результат работы целиком и полностью зависит от личных качеств отдельных сотрудников. При увольнении таких сотрудников проект останавливается.
- **Уровень 2 (Повторяемый).** Для перехода на **повторяемый** уровень необходимо внедрить формальные процедуры для выполнения основных элементов процесса конструирования. Результаты выполнения процесса соответствуют заданным требованиям и стандартам. Основное отличие от уровня 1 состоит в том, что выполнение процесса планируется и контролируется. Применяемые средства планирования и управления дают возможность повторения ранее достигнутых успехов.
- **Уровень 3** (**Определенный)** требует, чтобы все элементы процесса были определены, стандартизованы и задокументированы. Основное отличие от уровня 2 заключается в том, что элементы процесса уровня 3 планируются и управляются на основе единого стандарта компании. Качество разрабатываемого ПО уже не зависит от способностей отдельных личностей.

- **Уровень 4 (Управляемый).** С переходом на **управляемый** уровень в компании принимаются количественные показатели качества как программных продуктов, так и процесса. Это обеспечивает более точное планирование проекта и контроль качества его результатов. Основное отличие от уровня 3 состоит в более объективной, количественной оценке продукта и процесса.
- **Уровень 5** (**Оптимизирующий).** Высший уровень подразумевает, что главной задачей компании становится постоянное улучшение и повышение эффективности существующих процессов, ввод новых технологий. Основное отличие от уровня 4 заключается в том, что технология создания и сопровождения программных продуктов планомерно и последовательно совершенствуется.
- Каждый уровень СММ характеризуется *областью ключевых процессов* (ОКП), причем считается, что каждый последующий уровень включает в себя все характеристики предыдущих уровней.
- Область ключевых процессов образуют процессы, которые при совместном выполнении приводят к достижению определенного набора целей.
- **Пример,**
- ОКП 5-го уровня образуют процессы:
  - предотвращения дефектов;
  - управления изменениями технологии;
  - управления изменениями процесса.
- *Если все цели ОКП достигнуты, компании присваивается сертификат*

Управление проектами. Определения и концепции

- Управление проектами, лишь одна из 17 областей знаний программной инженерии, и то вспомогательная. Однако основной причиной большинства провалов программных проектов является именно применение неадекватных методов управления разработкой.
- *Проект* – это комплекс усилий, предпринимаемых с целью получения конкретных уникальных результатов в рамках отведенного времени и в пределах утвержденного бюджета, который выделяется на оплату ресурсов, используемых или потребляемых в ходе проекта. [Арчибальд Р.].
- *Проект,* есть комплекс действий, направленных на получение уникального результата, будь то продукт или услуга.
- *Цель проекта* описывает какие задачи должны быть решены в результате проекта.
- *Содержание проекта* - что именно является результатом проекта. Для информационной системы – ее функциональность.
- *Управление проектом определяет* “как”, с помощью каких действий, будет достигнута цель проекта и создан необходимый результат. При этом, управление проектами может и должно применяться на всех этапах жизненного цикла проекта, т.е. управление проектом есть постоянная деятельность.

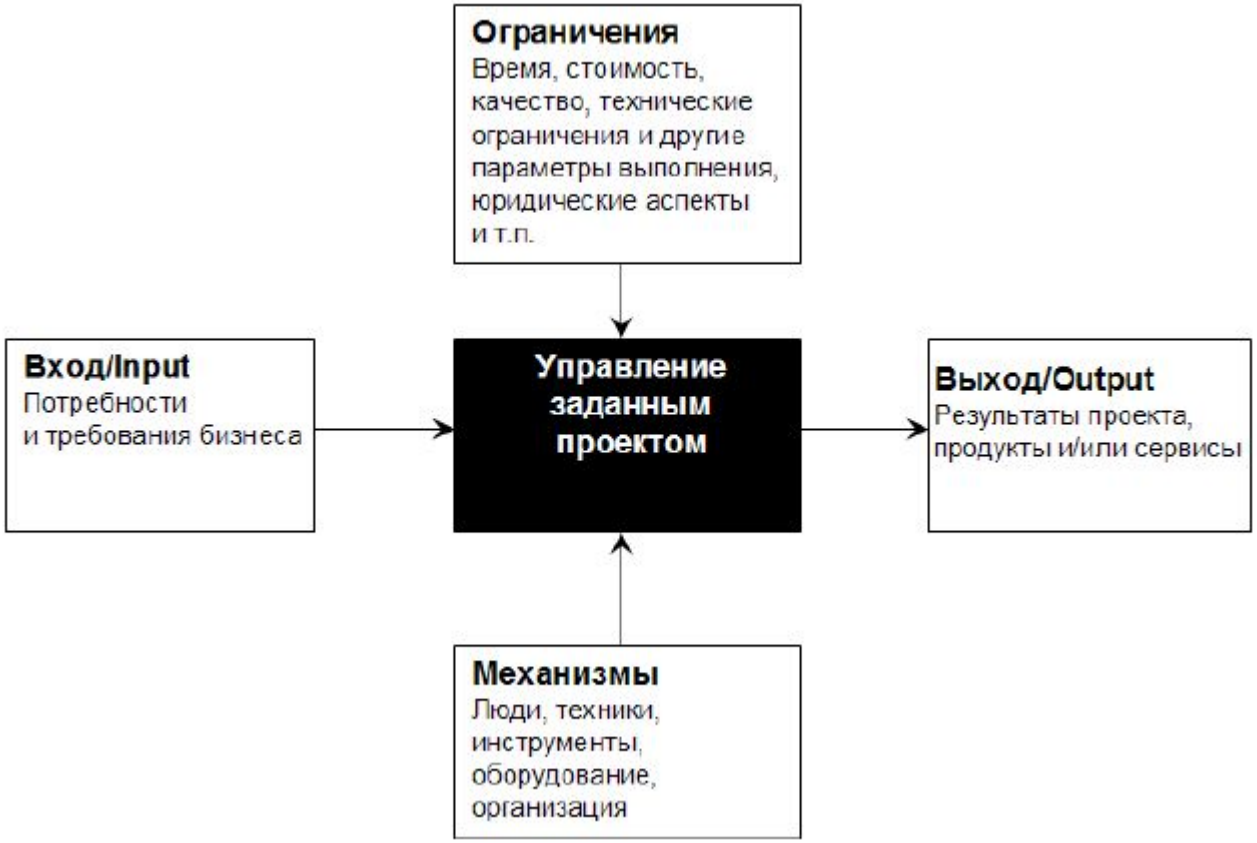
- В силу масштабности содержания проекта либо, например, разнородности его составных частей (например, программно-аппаратный комплекс шифрования данных) проект может быть разбит на несколько более мелких проектов.
- Так как с любым проектом ассоциированы цели, ресурсы, время, мы можем сформулировать, что управление проектами – есть дисциплина применения методов, практик и опыта к проектной деятельности для достижения целей проектов, при условии удовлетворения ограничений, определяющих их рамки.

Ограничения (constraints) в проектах.

- Чаще всего говорят о трех основных ограничениях или “железном треугольнике”
  1. Содержании проекта
  2. Времени
  3. Стоимости

- Любая оценка качества должна базироваться на измерениях и количественно выражаемых результатах измерений. измерений. Требования к качеству также должны описываться исчисляемыми характеристиками.

Процесс управления проектом. Роль ограничений.



- В приложении к индустрии программного обеспечения обычно добавляют четвертое ограничение – качество (quality), приемлемое качество.
- В зависимости от контекста и обсуждаемых в конкретном случае критериев качества, “приемлемое” качество может рассматриваться как необходимое, например, заданное требованиями качества и внутрикорпоративными стандартами.