

БАЗОВЫЕ ПОНЯТИЯ ПРОГРАММНОЙ ИНЖЕНЕРИИ

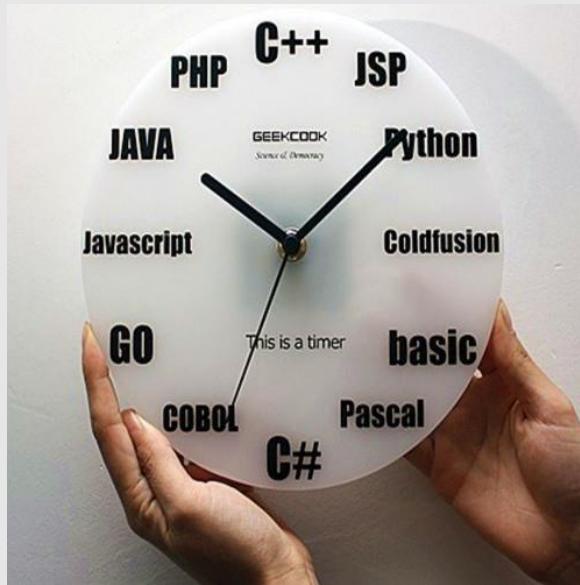
Машины должны работать.

Люди должны думать.

Девиз компании IBM

План лекции

1. История развития программной инженерии
2. Основные понятия и определения
3. Руководство к Своду Знаний по Программной Инженерии



1. История развития

Первый этап – «стихийное» программирование (60-е годы XX века)



1. История развития

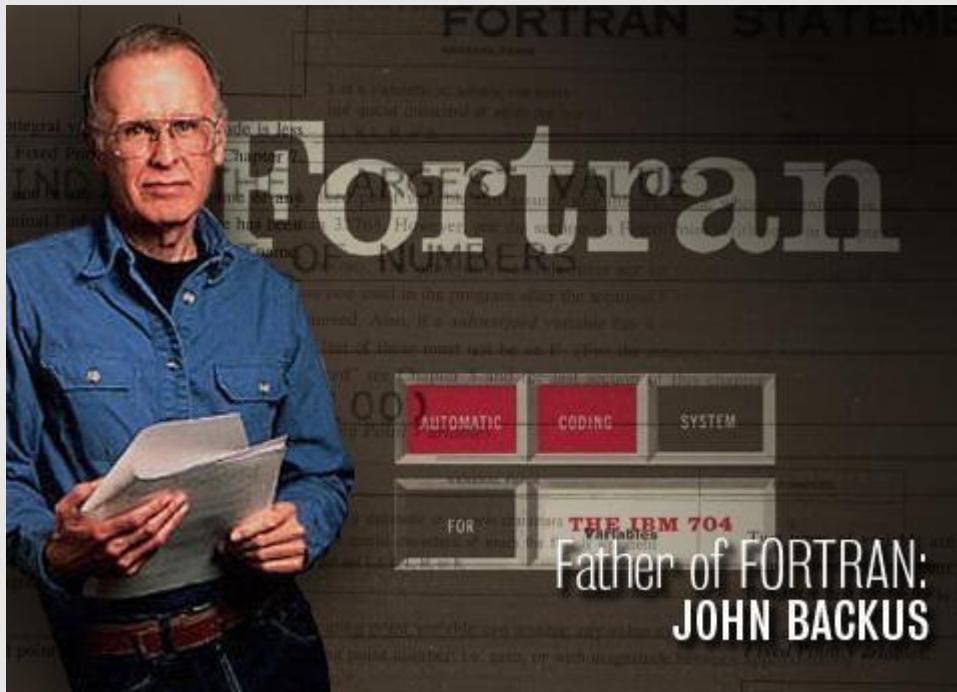
Первый этап – «стихийное» программирование

(60-е годы XX века)

- Программирование = искусство
- Машины языки → ассемблеры → алгоритмические языки (Fortran, Algol)
- Концепция повторного использования подпрограмм
- Решение, в основном, научно-технических задач
- Наличие достаточно четко сформулированного технического задания
- Отсутствие этапа проектирования.
- Составление документации после завершения разработки.
- Разработка "снизу вверх"
- Итог – **кризис** программирования

Языки программирования

- Фортран (FORmula TRANslator) (1954) – первый высокоуровневый язык программирования



- Сотрудник IBM
Джон Бэкус
(1924 – 2007)
Лауреат премии
Тьюринга

1. ИСТОРИЯ РАЗВИТИЯ

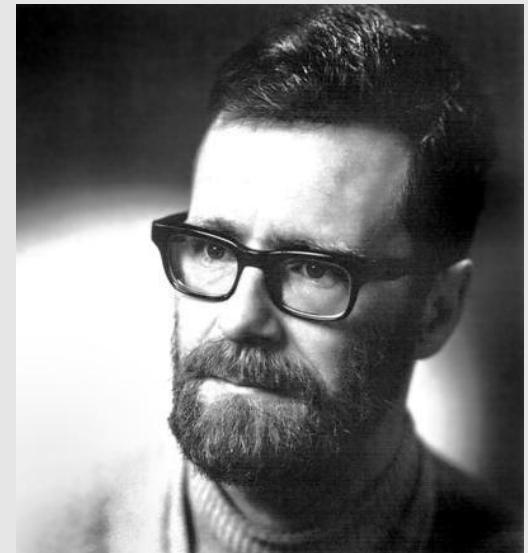
Второй этап: структурный подход (70-е годы XX века)

- Нисходящее функциональное проектирование («сверху вниз»)
- Декомпозиция (разбиение на части) – представление задачи в виде иерархии подзадач с целью последующей реализации в виде отдельных небольших подпрограмм
- Метод проектирования алгоритмов – метод пошаговой детализации
- Структурное кодирование без goto

1. ИСТОРИЯ РАЗВИТИЯ

Этапы: структурный подход (70-е годы XX века)

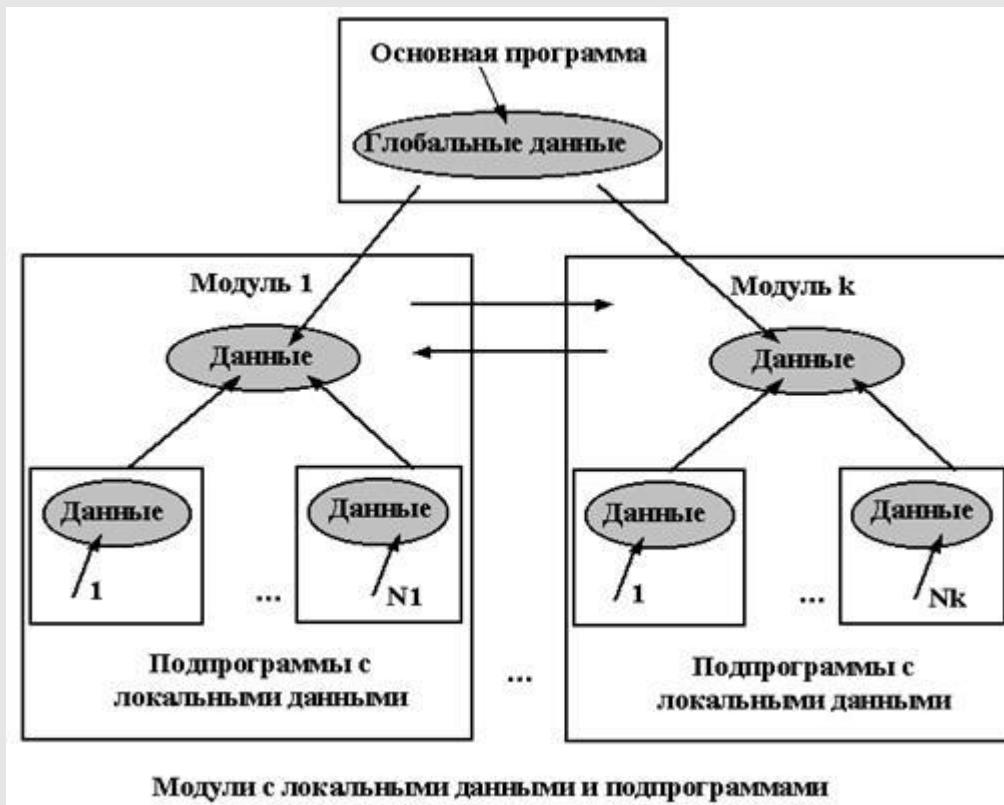
- «О вреде использования операторов GOTO» (GOTO statement considered harmful, 1968)
- Для записи любой программы в принципе достаточно только трех конструкций управления – последовательного выполнения, ветвления и цикла. То есть теоретически необходимость в использовании операторов перехода отсутствует.



Эдсгер Вибе Дейкстра

1. ИСТОРИЯ РАЗВИТИЯ

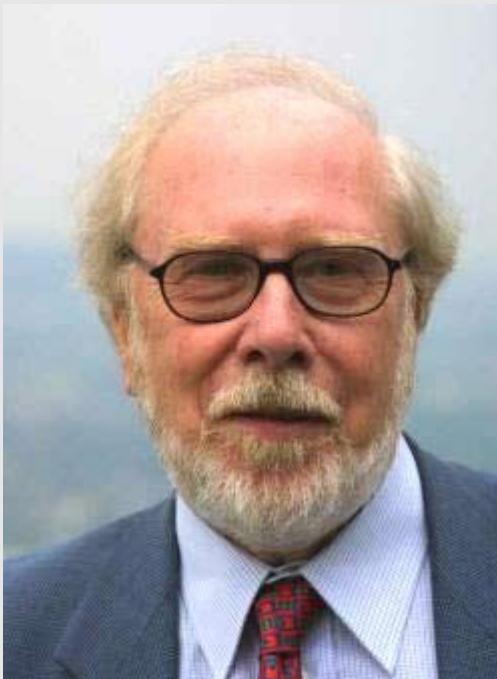
Структурный подход к программированию



- ✓ Pascal
- ✓ C

Языки программирования

Pascal (1970) - язык высокого уровня (ЯВУ) программирования общего назначения.



- Профессор Высшей политехнической школы (Цюрих) Никлаус Вирт (1934)
Лауреат премии Тьюринга

Языки программирования

- С - универсальный язык программирования высокого уровня, который считается языком системного программирования.



Разработан в середине 70-х годов Денисом Ритчи и Кеном Томпсоном. Лауреаты премии Тьюринга.

1. ИСТОРИЯ РАЗВИТИЯ

- **Технология модульного программирования** – выделение групп подпрограмм, использующих одни и те же глобальные данные, в отдельно компилируемые модули.
- Структурный подход + модульное программирование = **надежные программы**, размер которых не превышает 100 000 операторов.
- **Узким местом** модульного программирования стали межмодульные интерфейсы, ошибки в которых трудно обнаружить по причине раздельной компиляции модулей (ошибки выявляются только при выполнении программы).

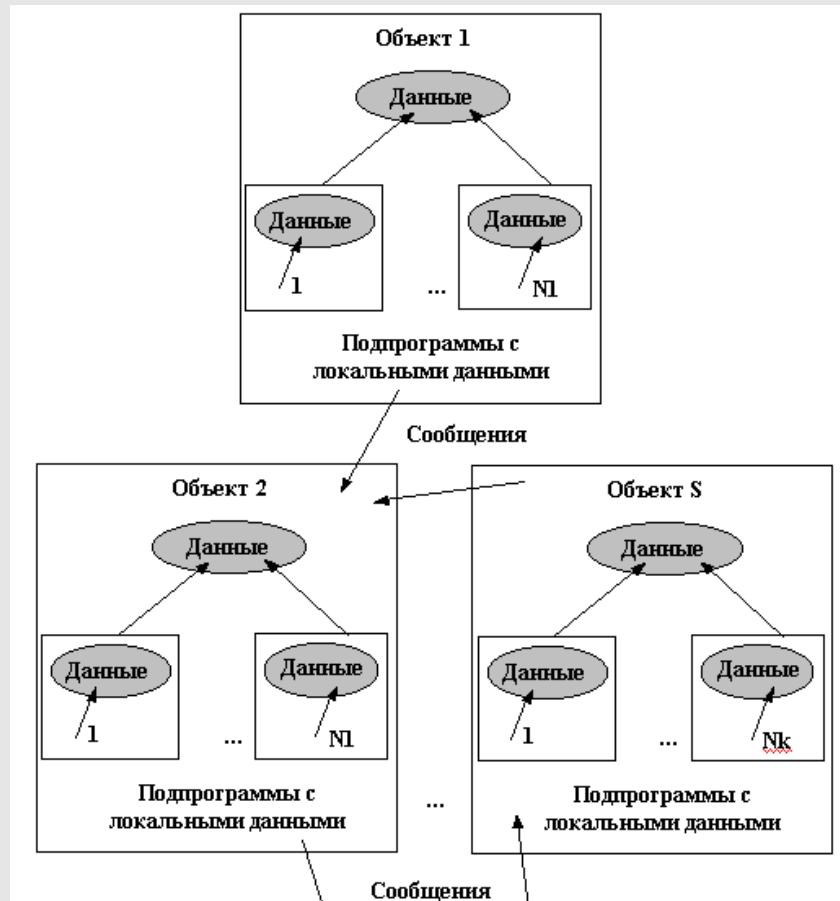
1. ИСТОРИЯ РАЗВИТИЯ

Этапы развития: объектный подход
80-90 годы

Программы = совокупность объектов, каждый из которых является экземпляром определенного типа (класса), а классы образуют иерархию с наследованием свойств. Взаимодействие программных объектов осуществляется путем передачи сообщений.

1. ИСТОРИЯ РАЗВИТИЯ

Этапы развития: объектный подход



- Создание сред **визуального программирования**
- Языки визуального объектно-ориентированного программирования (Delphi, Visual C++, C# и т. д.)

1. ИСТОРИЯ РАЗВИТИЯ

Этапы развития: объектный подход 80-90 годы

Недостатки

- противоречивые и переусложненные паттерны
- сложность и неоднозначность ООП-декомпозиции
- разработка программного обеспечения с использованием средств и возможностей *одного* языка программирования высокого уровня *одного* компилятора
- изменение реализации одного из программных объектов связано с перекомпоновкой всего ПО

1. История развития

Этапы развития: компонентный подход и case-технологии (с середины 90-х)

- Построение программного обеспечения из отдельных компонентов физически отдельно существующих частей программного обеспечения, которые взаимодействуют между собой через *стандартизованные двоичные интерфейсы*.
- Объекты-компоненты можно собрать в динамически вызываемые библиотеки или исполняемые файлы, распространять в двоичном виде (без исходных текстов) и использовать в любом языке программирования, поддерживающем соответствующую технологию.

1. История развития

Кризис в конце 70-х годов:

- Не соблюдались сроки
- Программный продукт (ПП) устаревал раньше, чем был готов
- Увеличивалась стоимость ПП

Необходимость:

- Контролировать процесс разработки ПС
- Прогнозировать и гарантировать стоимость разработки
- Соблюдать сроки разработки
- Гарантировать качество результатов

1. История развития

- Термин “Программная инженерия” (software engineering) появился впервые в октябре 1968 г. на конференции подкомитета НАТО по науке и технике (г.Гармиш, Германия). На конференции рассматривались проблемы проектирования, разработки, распространения и поддержки ПО.
- Термин «**программная инженерия**» прозвучал как некоторая дисциплина, которую надо создавать и которой надо руководствоваться при решении перечисленных проблем
- **Программная инженерия** (или **технология разработки ПО**) возникла и развивалась под давлением роста стоимости создаваемого ПО.

Фундаментальная идея

“Разработка ПО является формальным процессом, который можно изучать и совершенствовать.”

Освоение и правильное использование методов и средств создания ПО позволяют повысить качество , обеспечить управляемость процесса разработки ПО и увеличить срок его жизни.

Факторы:

1. Накопление в области создания ПО значительного объема разнородных знаний
2. Появление новых методов проектирования и разработки ПО
3. Совершенствование методов обнаружения ошибок в ПО
4. Эффективная организация работы коллективов разработчиков ПО

2. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Программная инженерия -

применение систематизированного, дисциплинированного и оцениваемого по количественным параметрам подхода к разработке, функционированию и сопровождению программного обеспечения, то есть применение инженерного подхода к созданию ПО

IEEE Computer Society

- ведущая организация в области компьютерных наук, крупнейшее профессиональное сообщество, объединяющее более 100 тыс. специалистов, работающих в области компьютерных наук и технологий.

2. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

- **Программное обеспечение** (software) – совокупность компьютерных программ и необходимых для их эксплуатации документов
- **Программное средство** – ограниченная часть программного обеспечения системы обработки информации, имеющая определенное функциональное назначение
- **Программный продукт** (software product) – совокупность компьютерных программ, процедур и связанных с ними документации и данных, предназначенная для удовлетворения потребностей пользователей, широкого распространения и продажи.

2. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

- **Разработка ПО** – совокупность процессов, обеспечивающих создание и развитие программного обеспечения.
- **Метод** представляет собой совокупность следующих составляющих:
 - концепций и теоретических основ:
 - структурный,
 - объектно-ориентированный ,
 - компонентный
 - подходы к разработке ПО;

2. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

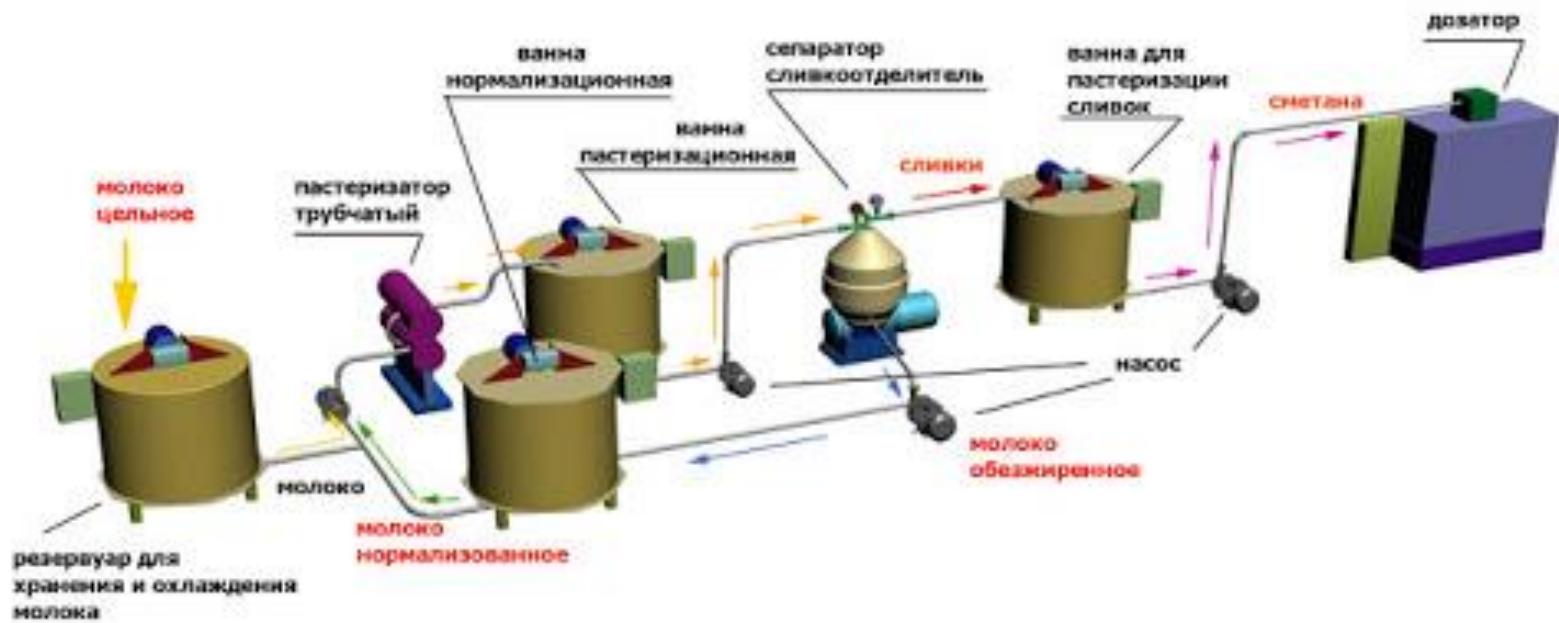
- нотаций, используемых для построения моделей статической структуры и динамики поведения проектируемой системы;
- процедур, определяющих практическое применение метода: последовательность и правила построения моделей, критерии, используемые для оценки результатов.
- **Методология разработки ПО** – система принципов и способов организации процесса разработки программ
- **Инструментальные средства** – совокупность программ и программных комплексов, обеспечивающих технологию разработки, отладки и внедрения создаваемых программных продуктов.

2. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

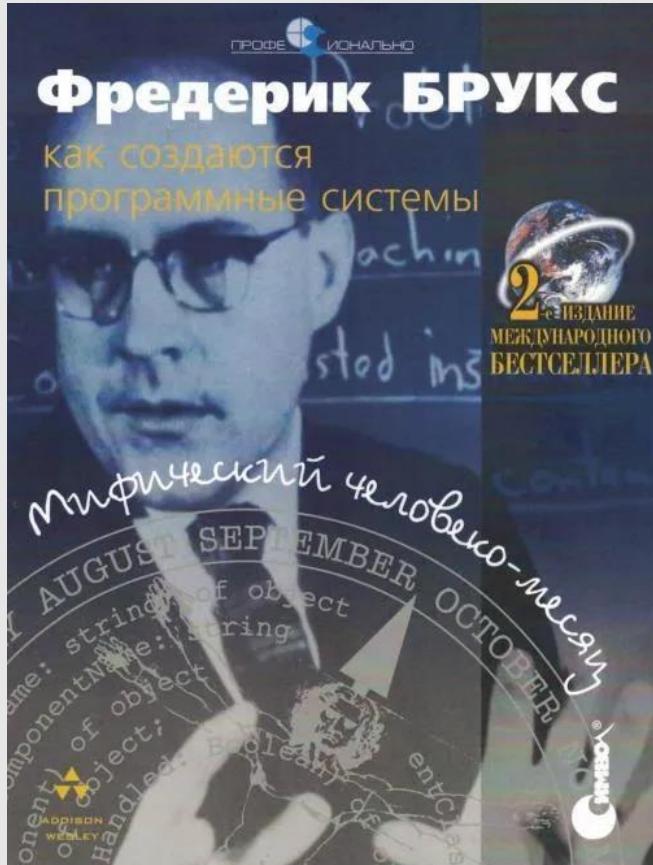


Изучает вопросы построения компьютерных программ, обобщает опыт программирования в виде комплекса общих знаний и правил регламентации инженерной деятельности разработчиков программного обеспечения

3. ОСОБЕННОСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ



3. ОСОБЕННОСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ



«Программист подобно поэту, работает почти непосредственно с чистой мыслью. Он строит свои замки в воздухе и из воздуха, творя силой воображения. Однако эта податливость «материала» таит свои проблемы. Во-первых, необходима безошибочная точность действий: одна ошибка и чудо не состоялось!»

Ф. Брукс

3. ОСОБЕННОСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1. Разработка ПО носит творческий характер
2. ПО представляет собой некоторую совокупность текстов (т.е. статических объектов), смысл же (семантика) этих текстов выражается процессами обработки данных и действиями пользователей, запускающих эти процессы (т.е. является динамическим)
3. ПО при своём использовании (эксплуатации) не расходуется и не расходует используемых ресурсов
4. В процессе эксплуатации в ПО могут быть внесены изменения и оно может быть доработано
5. Массовое производство программного продукта имеет очень низкую стоимость, его экономическая модель существенно отличается от модели обычных товаров

3. ОСОБЕННОСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИТ-ПРОЕКТ

Проект имеет конкретную ограниченную во времени цель

Выполнение проекта ограничено четкими временными рамками

Каждый проект по-своему неповторим (уникален)

На выполнение проекта выделено ограниченное количество ресурсов

На время выполнения проекта создаются специальные проектные команды

3. ОСОБЕННОСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИТ-ПРОЕКТ

- Успешные проекты – завершены вовремя в полном объеме и в рамках бюджета;
- Спорные проекты – завершены не в полном объеме и/или с нарушением сроков, и/или перерасходом бюджета;
- Проваленные проекты – не доведены до конца (перерасход средств, низкое качество)

	2004	2006	2008	2010	2012	2013
УСПЕШНЫЕ	29%	35%	32%	37%	39%	36%
ПРОВАЛЬНЫЕ	18%	19%	24%	21%	18%	16%
СПОРНЫЕ	53%	46%	44%	42%	43%	48%

3. ОСОБЕННОСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

ИТ-ПРОЕКТ

Основными причинами неудачи ИТ-проектов признаются:

- Нереалистичные временные рамки проекта
- Недостаток количества исполнителей
- Размытые границы проекта
- Недостаток финансирования
- Нехватка квалифицированных кадров

4. СТАНДАРТЫ

Стандарт (standard) - норма, образец, мерило:

- утверждаемый компетентным органом нормативно-технический документ, устанавливающий комплекс норм и правил по отношению к объекту стандартизации
- типовой образец, эталон, модель, принимаемые за исходные для сопоставления с ними других предметов

ОСНОВНЫЕ ТИПЫ СТАНДАРТОВ

- **Корпоративные стандарты** разрабатываются крупными фирмами с целью повышения качества своей продукции. Не сертифицируются, но являются обязательными для применения внутри корпорации
- **Отраслевые стандарты** действуют в пределах организаций некоторой отрасли (министерства). Разрабатываются с учетом требований мирового опыта и специфики отрасли. Подлежат сертификации

ОСНОВНЫЕ ТИПЫ СТАНДАРТОВ

- **Государственные стандарты** (ГОСТы) принимаются государственными органами и имеют силу закона. Разрабатываются с учетом мирового опыта или на основе отраслевых стандартов. Могут иметь как рекомендательный, так и обязательный характер. Для сертификации создаются государственные или лицензированные органы сертификации
- **Международные стандарты** разрабатываются специальными международными организациями на основе мирового опыта и лучших корпоративных стандартов. Имеют сугубо рекомендательный характер

КЛАССИФИКАЦИЯ

Классификация по объекту стандартизации:

1. Стандарты, определяющие требования к качеству ПО.
2. Стандарты, определяющие требования к документации.
3. Стандарты, определяющие термины по программному обеспечению.
4. Стандарты на процессы жизненного цикла программного обеспечения.
5. Обучающие стандарты.

4. РАЗРАБОТЧИКИ СТАНДАРТОВ

- **ISO** - The International Standards Organization - международная организация по стандартизации, работающая в сотрудничестве с IEC - The International Electrotechnical Commission - международной электротехнической комиссией
- **IEEE Computer Society** - профессиональное объединение специалистов в области программной инженерии
- **ACM** - Association for Computing Machinery – Ассоциация по вычислительной технике

4. SOFTWARE ENGINEERING BODY OF KNOWLEDGE

- **SWEBOK** – Software Engineering Body of Knowledge – Свод знаний по программной инженерии – содержит описания состава знаний по разделам (областям знаний) программной инженерии.
- **ACM/IEEE CC 2001** – Computing Curricula 2001 – Академический образовательный стандарт в области компьютерных наук. Выделены 4 основных раздела компьютерных наук:
 - *Computer science,*
 - *Computer engineering,*
 - *Software engineering,*
 - *Information systems,*

по каждому из которых описаны области знаний соответствующего раздела, состав и планы рекомендуемых курсов.

4. ОБЛАСТИ ЗНАНИЙ SWEBOK

Guide to the Software Engineering Body of Knowledge (SWEBOk), IEEE 2004 Version- Руководство к Своду Знаний по Программной Инженерии [SWEBOk, 2004]

- Ядру знаний SWEBOk соответствует стандарт ISO/IEC TR 19759:2005.
- **Версии:**
 - 2004 г. (*SWEBOk V2*) - десять областей знаний (5 основных и 5 областей управления);
 - 2013 г. (*SWEBOk V3*) - пятнадцать областей (+ теоретические основы ПИ, экономика и описание профессиональных навыков по ПИ)

SWEBOK

Guide to the

Software Engineering Body of Knowledge

2004 Version

Executive Editors

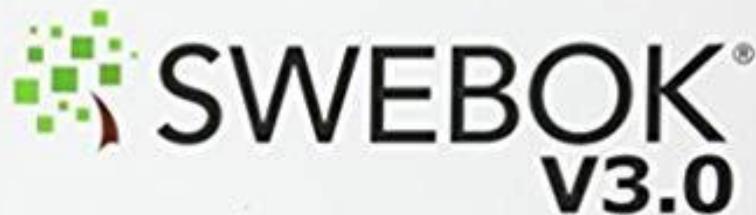
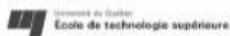
Alain Abran, École de technologie supérieure
James W. Moore, The MITRE Corp.

Editors

Pierre Bourque, École de technologie supérieure
Robert Dupuis, Université du Québec à Montréal



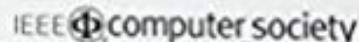
Project managed by:



*Guide to the Software
Engineering Body of Knowledge*

Editors

Pierre Bourque
Richard E. (Dick) Fairley

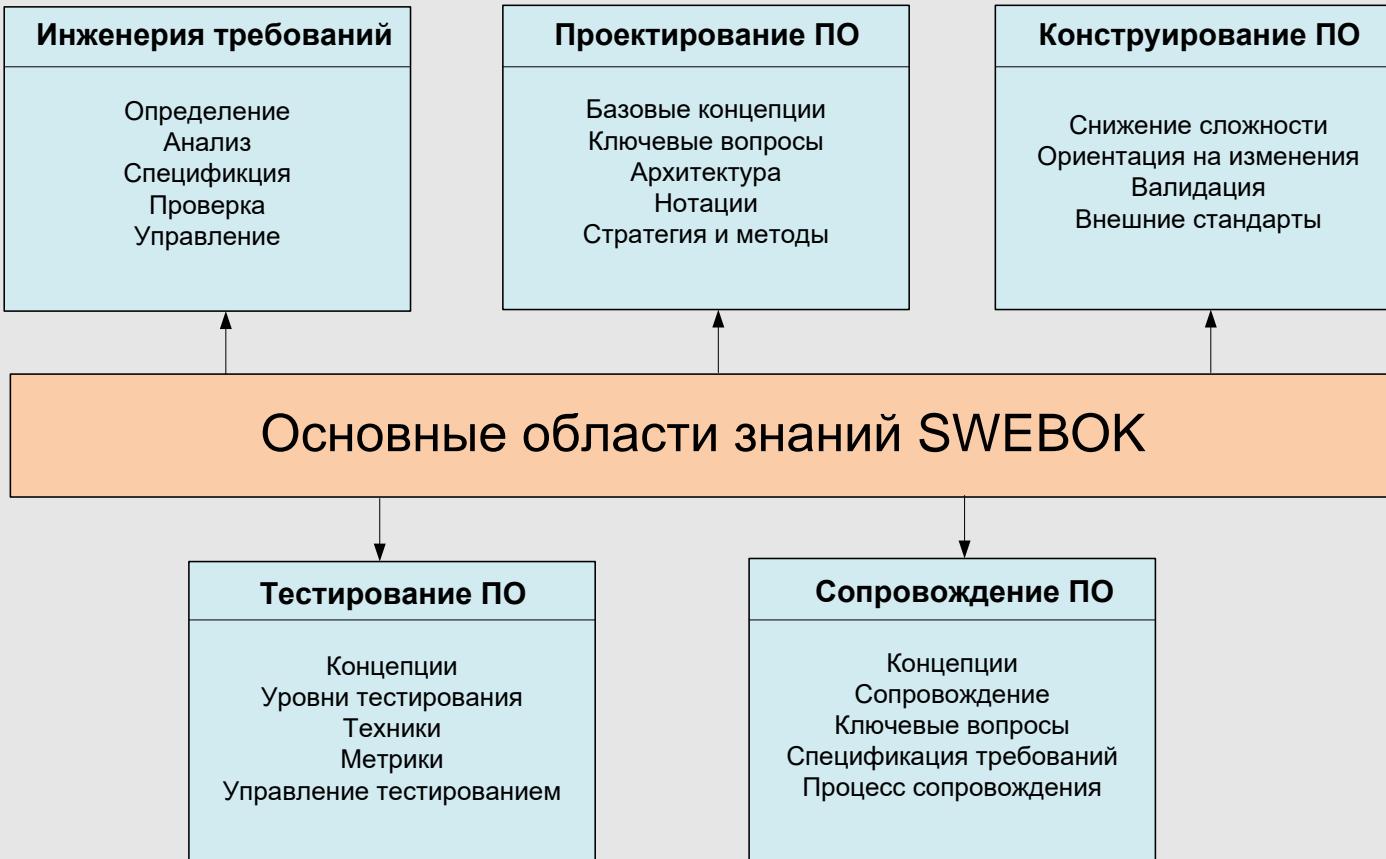


4. ОБЛАСТИ ЗНАНИЙ SWEBOK

SWEBOK также включает обзор смежных дисциплин связь с которыми представлена как фундаментальная, важная и обоснованная для программной инженерии:

1. Computer engineering – разработка компьютеров.
2. Computer science – информатика.
3. Management – общий менеджмент.
4. Mathematics – математика.
5. Project management – управление проектами.
6. Quality management – управление качеством.
7. Systems engineering – системное проектирование.

Основные области знаний SWEBOK



ТРЕБОВАНИЯ К ПО

Требования к продукту и процессу

Условия выполнения и режим работы ПО, ограничения на среду исполнения; определение принципов взаимодействия с другими программами.

Функциональные требования определяют назначение и функции системы.

Нефункциональные требования определяют условия исполнения ПО, переносимости и доступа к данным.

Системные требования - требования к программной системе в целом.

ТРЕБОВАНИЯ К ПО

Определение

Инженерия требований (англ. *requirements engineering*)

- процесс формулировки, документирования и поддержки требований к ПО, а также соответствующая область программной инженерии.

определение

(*elicitation*)

анализ

(*analysis*)

спецификация

(*specification*)

проверка

(*validation*)

управление

(*management*)

Интеграция требований во все процессы ЖЦ;

|

контроль реализации требований;

|

необходимая корректировка требований.

ПРОЕКТИРОВАНИЕ ПО

Определение

Проектирование ПО (англ. *software design*) - процесс определения архитектуры ПО, набора составляющих компонентов и их интерфейсов, прочих характеристик системы и конечного состава программного продукта.

Архитектура программного обеспечения - высокоуровневое представление структуры системы и спецификация ее компонентов и логики их взаимодействия.

КОНСТРУИРОВАНИЕ ПО

Определение

Конструирование ПО (англ. *software construction*) – создание ПО из составных элементов (блоков, операторов, функций) и его проверка методами тестирования.

Тестирование ПО - это процесс проверки готовой программы в *статике* (обзоры кода, инспекции и т. п.) и *динамике* (прогон программы на тестовых данных) с целью обеспечить ее соответствие заданным требованиям.

Сопровождение ПО (англ. *software maintenance*) - совокупность действий по обеспечению работы ПО, внесению изменений при выявлении ошибок, адаптации к новой среде исполнения, улучшения производительности или других характеристик ПО.

4. ОБЛАСТИ УПРАВЛЕНИЯ SWEBOK



4. ОБЛАСТИ УПРАВЛЕНИЯ SWEBOK

Управление конфигурацией – дисциплина идентификации компонентов системы, определения функциональных и физических характеристик аппаратного и программного обеспечения для проведения контроля внесения изменений и трассирования конфигурации на протяжении ЖЦ.

Управление инженерией ПО (менеджмент) – руководство работами команды разработчиков ПО в процессе выполнения плана проекта, определение критериев и оценка процессов и продуктов проекта с использованием общих методов управления, планирования и контроля работ.

4. ОБЛАСТИ УПРАВЛЕНИЯ SWEBOK

- **Процесс инженерии ПО** включает концепции, инфраструктуру, методы определения и измерения этапов ЖЦ, поиск ошибок и внесение изменений, также анализ и оценку качества продукта.

Качество ПО – набор характеристик продукта или сервиса, которые характеризуют его способность удовлетворить установленным или предполагаемым потребностям заказчика.

ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

ЛЕКЦИЯ № 2

*Ходить по воде и разрабатывать программы
согласно ТЗ очень просто, если они заморожены*

ПЛАН ЛЕКЦИИ

1. Жизненный цикл (ЖЦ) программного обеспечения (ПО)
2. Стратегии разработки ПО
3. Модели ЖЦ ПО
4. Спиральная модель

1. ЖИЗНЕННЫЙ ЦИКЛ ПО

– это непрерывный процесс (совокупность процессов), который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации

Процесс – совокупность взаимосвязанных или взаимодействующих видов деятельности (работ), преобразующих входы в выходы.

Выход процесса – наблюдаемый результат успешного достижения цели процесса:

- изготовление какого-либо артефакта;
- существенное изменение состояния;
- удовлетворение заданных ограничений, требований, конечных целей и т.п.

1. ЖИЗНЕННЫЙ ЦИКЛ ПО

В основе практически всех современных промышленных технологий создания ПС лежит международный стандарт **ISO/IEC 12207 «Системная и программная инженерия. Процессы жизненного цикла программных средств.»**

Стандарт ISO/IEC 12207 определяет организацию ЖЦ программного продукта как совокупность процессов, каждый из которых разбит на действия, состоящие из отдельных задач; устанавливает структуру (архитектуру) ЖЦ программного продукта в виде перечня процессов, действий и задач.

1. ЖИЗНЕННЫЙ ЦИКЛ ПО

Модель жизненного цикла – структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач, организуемых в стадии, выполняемых на протяжении ЖЦ.

Стадии (этапы) жизненного цикла :

- формирование требований на основе анализа предметной области
- проектирование
- реализация
- внедрение (ввод системы в эксплуатацию)
- эксплуатация (сопровождение проекта).

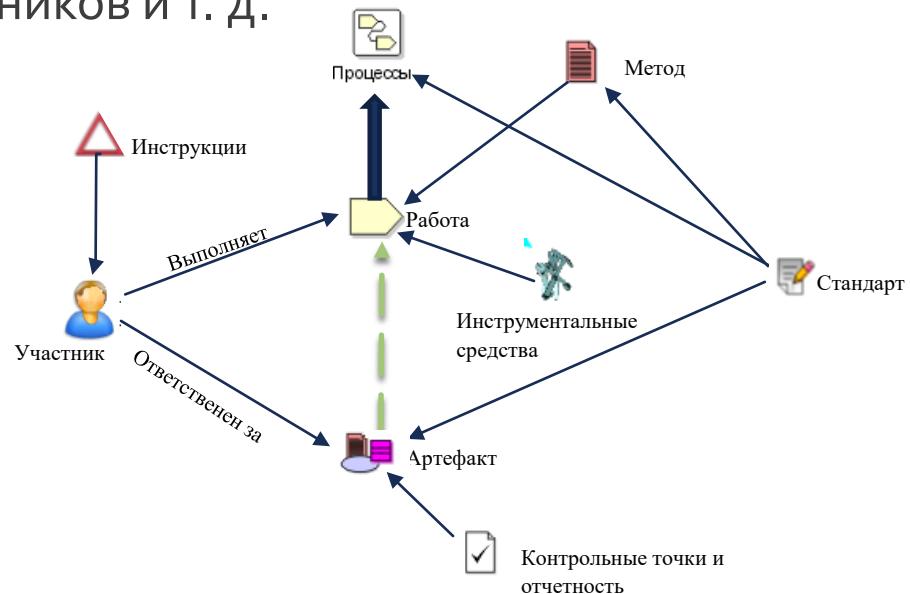
1. ЖИЗНЕННЫЙ ЦИКЛ ПО



1. ЖИЗНЕННЫЙ ЦИКЛ ПО

Для каждой стадии определяют:

- состав и последовательность выполняемых работ,
- получаемые результаты
- методы и средства, необходимые для выполнения работ,
- роли и ответственность участников и т. д.

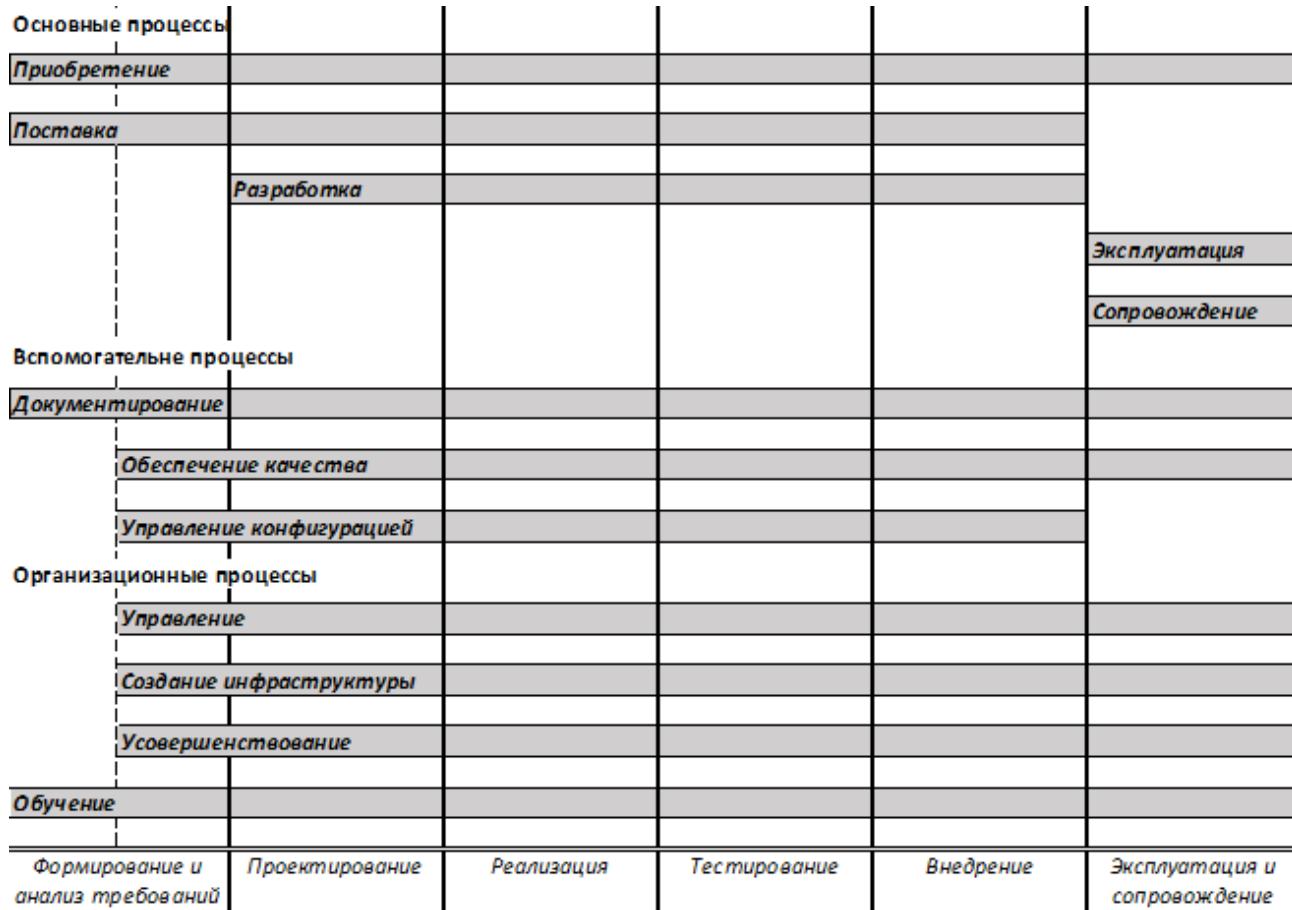


ISO/IEC/IEEE 12207:2017

«Системная и программная инженерия.
Процессы жизненного цикла программных
средств» (*Systems and software engineering –
Software life cycle processes*)

- Стандарт не требует использования какой-либо конкретной модели жизненного цикла.
- Стандарт требует, чтобы в каждом проекте определялась подходящая модель жизненного цикла.
- Стандарт не содержит требований использования какой-либо заданной совокупности стадий.
- Последовательность процессов не предполагает какой-либо зависимости от времени.
- Пользователь настоящего стандарта может самостоятельно выбирать и назначать процессы, виды деятельности и задачи как наиболее подходящие и эффективные.

1. ЖИЗНЕННЫЙ ЦИКЛ ПО



- Состав стадий и выполняемые на каждой стадии процессы зависят от модели ЖЦ ПО.

1. ЖИЗНЕННЫЙ ЦИКЛ ПО

основные

- приобретение (заказ)
- поставка
- разработка
- эксплуатация
- сопровождение

вспомогательные

- документирование
- управление конфигурацией
- обеспечение качества
- верификация
- валидация (аттестация)
- оценка (совместный просмотр)
- аудит
- решение проблем

организационные

- управление проектами
- создание и сопровождение инфраструктуры проекта
- усовершенствование (определение, оценка и улучшение самого ЖЦП)
- обучение

1. ЖИЗНЕННЫЙ ЦИКЛ ПО

Основные:

- **Приобретение** (действия и задачи заказчика, приобретающего ПО)
- **Поставка** (действия и задачи поставщика, который снабжает заказчика программным продуктом или услугой)
- **Разработка** (действия и задачи, выполняемые разработчиком: создание ПО, оформление проектной и эксплуатационной документации, подготовка тестовых и учебных материалов и т. д.)
- **Эксплуатация и Сопровождение** — внесений изменений в ПО в целях исправления ошибок, повышения производительности или адаптации к изменившимся условиям работы или требованиям.

1. ЖИЗНЕННЫЙ ЦИКЛ ПО

Вспомогательные

- **Документирование** (формализованное описание информации, созданной в течение ЖЦ ПО)
- **Управление конфигурацией** (применение административных и технических процедур на всем протяжении ЖЦ ПО для определения состояния компонентов ПО, управления его модификациями).
- **Обеспечение качества** (обеспечение гарантий того, что ИС и процессы ее ЖЦ соответствуют заданным требованиям и утвержденным планам)
- **Верификация** (определение того, что программные продукты, являющиеся результатами некоторого действия, полностью удовлетворяют требованиям или условиям, обусловленным предшествующими действиями)

1. ЖИЗНЕННЫЙ ЦИКЛ ПО

Вспомогательные

- **Аттестация** (определение полноты соответствия заданных требований и созданной системы их конкретному функциональному назначению).
- **Совместная оценка** (оценка состояния работ по проекту: контроль планирования и управления ресурсами, персоналом, аппаратурой, инструментальными средствами)
- **Аудит** (определение соответствия требованиям, планам и условиям договора)
- **Разрешение проблем** (анализ и решение проблем, независимо от их происхождения или источника, которые обнаружены в ходе разработки, эксплуатации, сопровождения или других процессов)

1. ЖИЗНЕННЫЙ ЦИКЛ ПО

Организационные

- **Управление** (действия и задачи, которые могут выполняться любой стороной, управляющей своими процессами)
- **Создание инфраструктуры** (выбор и сопровождение технологии, стандартов и инструментальных средств, выбор и установка аппаратных и программных средств, используемых для разработки, эксплуатации или сопровождения ПО)
- **Усовершенствование** (оценка, измерение, контроль и усовершенствование процессов ЖЦ)
- **Обучение** (первоначальное обучение и последующее постоянное повышение квалификации персонала)

ТАБЛИЦА. СОДЕРЖАНИЕ ОСНОВНЫХ ПРОЦЕССОВ ЖЦ ПО ИС (ISO/IEC 12207)

Процесс (Исполнитель процесса)	Действия	Вход	Результат
Приобретение (заказчик)	<ul style="list-style-type: none">· Инициирование· Подготовка заявочных предложений· Подготовка договора· Контроль деятельности поставщика· Приемка ИС	<ul style="list-style-type: none">· Решение о начале работ по внедрению ИС· Результаты обследования деятельности заказчика· Результаты анализа рынка ИС/ тендера· План поставки/ разработки· Комплексный тест ИС	<ul style="list-style-type: none">· Технико-экономическое обоснование внедрения ИС· Техническое задание на ИС· Договор на поставку/ разработку· Акты приемки этапов работы· Акт приемно-сдаточных испытаний

ТАБЛИЦА. СОДЕРЖАНИЕ ОСНОВНЫХ ПРОЦЕССОВ ЖЦ ПО ИС (ISO/IEC 12207)

Процесс (Исполнитель процесса)	Действия	Вход	Результат
Поставка (разработчик ИС)	<ul style="list-style-type: none">· Инициирование· Ответ на заявочные предложения· Подготовка договора <p>Планирование исполнения</p> <ul style="list-style-type: none">· Поставка ИС	<ul style="list-style-type: none">· Техническое задание на ИС· Решение руководства об участии в разработке· Результаты тендера <p>· Техническое задание на ИС</p> <ul style="list-style-type: none">· План управления проектом· Разработанная ИС и документация	<ul style="list-style-type: none">· Решение об участии в разработке· Коммерческие предложения/ конкурсная заявка· Договор на поставку/ разработку <p>· План управления проектом</p> <ul style="list-style-type: none">· Реализация/ корректировка· Акт приемно- сдаточных испытаний

Процесс (Исполнитель процесса)	Действия	Вход	Результат
Разработка (разработчик ИС)	<ul style="list-style-type: none"> · Подготовка · Анализ требований к ИС · Проектирование архитектуры ИС · Разработка требований к ПО · Проектирование архитектуры ПО · Детальное проектирование ПО · Кодирование и тестирование ПО · Интеграция ПО и квалификационное тестирование ПО · Интеграция ИС и квалификационное тестирование ИС 	<ul style="list-style-type: none"> · Техническое задание на ИС · Техническое задание на ИС, модель ЖЦ · Подсистемы ИС · Спецификации требования к компонентам ПО · Архитектура ПО · Материалы детального проектирования ПО · План интеграции ПО, тесты · Архитектура ИС, ПО, документация на ИС, тесты 	<ul style="list-style-type: none"> · Используемая модель ЖЦ, стандарты разработки · План работ · Состав подсистем, компоненты оборудования · Спецификации требования к компонентам ПО · Состав компонентов ПО, интерфейсы с БД, план интеграции ПО · Проект БД, спецификации интерфейсов между компонентами ПО, требования к тестам · Тексты модулей ПО, акты автономного тестирования · Оценка соответствия комплекса ПО требованиям ТЗ · Оценка соответствия ПО, БД, технического комплекса и комплекта документации требованиям ТЗ

2. СТРАТЕГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

- **Стратегия** (*искусство полководца*) разработки ПО
 - порядок следования и содержания основных этапов разработки
- **Методология**
 - совокупность методов, правил и принципов управления процессом разработки, которые делают её более эффективной

2. БАЗОВЫЕ СТРАТЕГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

- Каскадная
- Инкрементная
- Эволюционная

Определяются характеристиками:

- проекта
- требований к программному продукту
- команды разработчиков
- команды пользователей.

КАСКАДНАЯ СТРАТЕГИЯ (1970, УИНСТОН РОЙС)

Характеристика:

- последовательное выполнение входящих в ее состав этапов;
- окончание каждого предыдущего этапа до начала следующего;
- отсутствие (или определенным ограничением) возврата к предыдущим этапам;
- наличие результата только в после завершения всех этапов.

КАСКАДНАЯ СТРАТЕГИЯ: ДОСТОИНСТВА

1. Простота применения стратегии
2. Простота планирования, контроля и управления проектом
3. Доступность для понимания заказчиками

КАСКАДНАЯ СТРАТЕГИЯ: НЕДОСТАТКИ

1. Сложность полного формулирования требований в начале процесса разработки и невозможность их динамического изменения на протяжении ЖЦ
2. Линейность структуры процесса разработки
3. Непригодность промежуточных продуктов для использования
4. Недостаточное участие пользователя в процессе разработки ПС – невозможность предварительной оценки пользователем качества программного средства.

КАСКАДНАЯ СТРАТЕГИЯ: ОБЛАСТЬ ПРИМЕНЕНИЯ

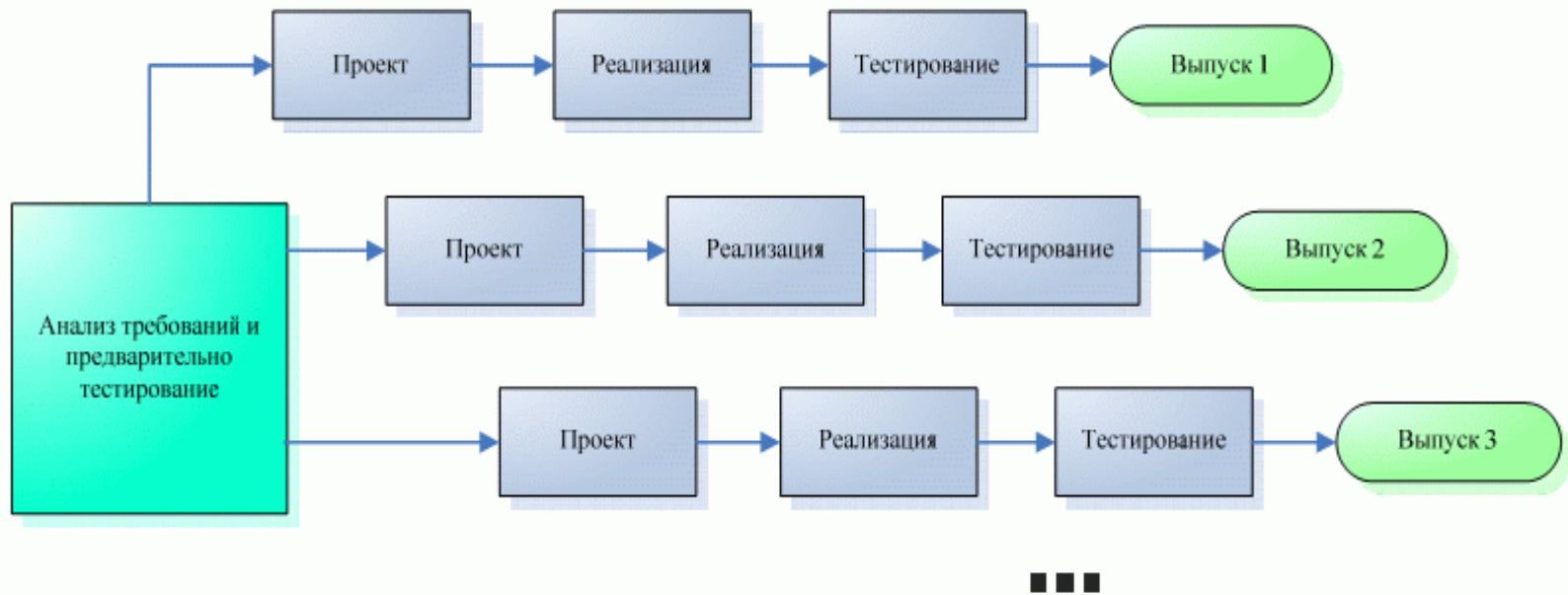
1. Проекты с четкими, неизменяемыми в течение ЖЦ требованиями и понятной реализацией.
2. Проекты невысокой сложности, например:
 - создание программного средства или системы такого же типа, как уже разрабатывались разработчиками;
 - создание новой версии уже существующего программного средства или системы;
 - перенос уже существующего продукта на новую платформу.
3. В качестве составной части моделей ЖЦ, реализующих другие стратегии разработки при выполнении больших проектов.

ИНКРЕМЕНТНАЯ СТРАТЕГИЯ: ХАРАКТЕРИСТИКА

- многократное выполнение этапов разработки с запланированным улучшением результата;
- полное определение всех требований к программному средству (системе) в начале процесса разработки;
- полный набор требований реализуется постепенно в соответствии с планом в последовательных циклах разработки.

Результат каждого цикла называется **инкрементом**.

ИНКРЕМЕНТНАЯ СТРАТЕГИЯ РАЗРАБОТКИ



ИНКРЕМЕНТНАЯ СТРАТЕГИЯ: ДОСТОИНСТВА

1. Возможность получения функционального продукта после реализации каждого инкремента
2. Короткая продолжительность создания инкремента
3. Предотвращение реализации громоздких спецификаций требований
4. Стабильность требований во время создания определенного инкремента
5. Возможность учета изменившихся требований
6. Снижение рисков по сравнению с каскадной стратегией
7. Включение в процесс пользователей, что позволяет оценить функциональные возможности продукта на более ранних этапах разработки и в конечном итоге приводит к повышению качества программного продукта, снижению затрат и времени на его разработку.

ИНКРЕМЕНТНАЯ СТРАТЕГИЯ: НЕДОСТАТКИ

1. Необходимость полного функционального определения системы или программного средства в начале ЖЦ для обеспечения планирования инкрементов и управления проектом
2. Возможность текущего изменения требований к системе или программному средству, которые уже реализованы в предыдущих инкрементах
3. Сложность планирования и распределения работ;
4. Проявление человеческого фактора, связанного с тенденцией к оттягиванию решения трудных проблем на поздние инкременты, что может нарушить график работ или снизить качество программного продукта.

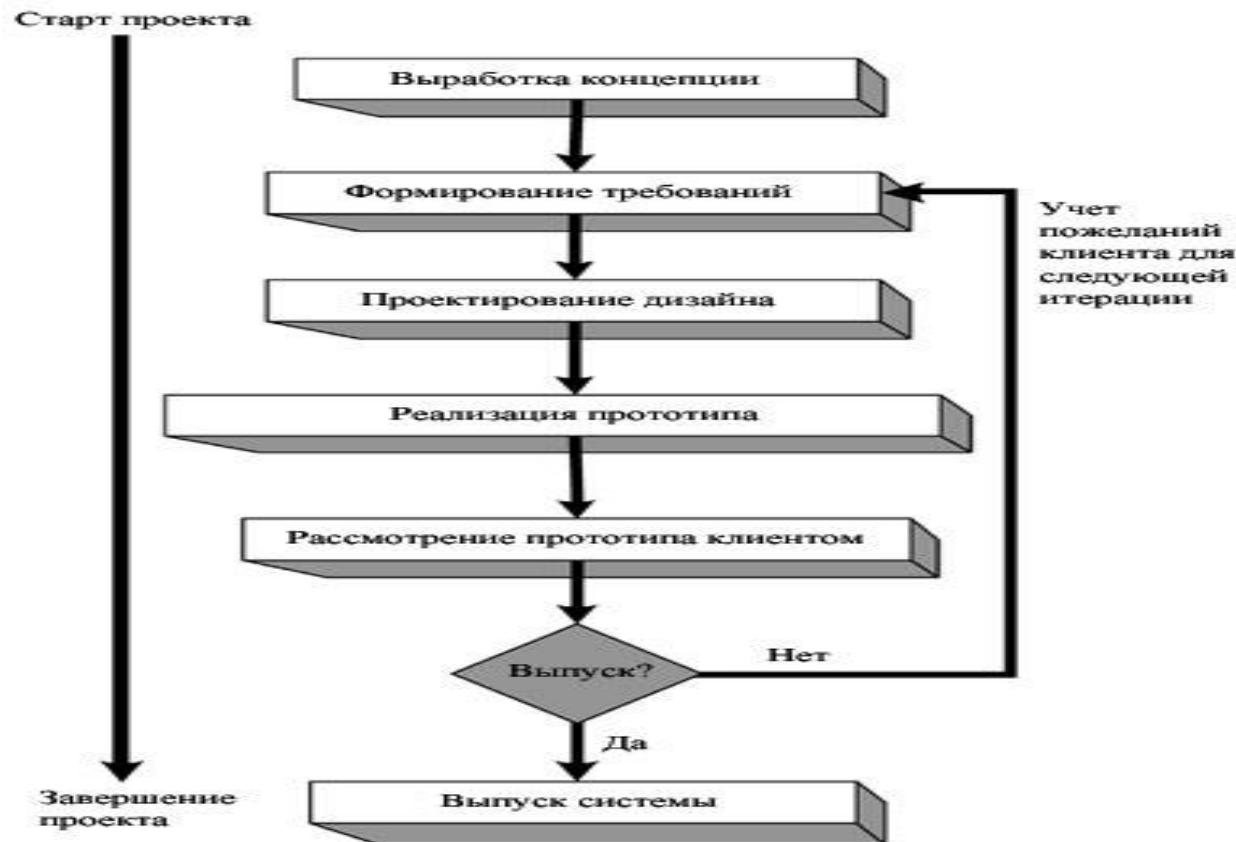
ИНКРЕМЕНТНАЯ СТРАТЕГИЯ: ОБЛАСТЬ ПРИМЕНЕНИЯ

1. Проекты, в которых большинство требований можно сформулировать заранее, но часть из них могут быть уточнены через определенный период времени
2. Сложные проекты с заранее сформулированными требованиями
3. При необходимости быстро поставить на рынок продукт, имеющий базовые функциональные свойства
4. Проекты с низкой или средней степенью рисков.

ЭВОЛЮЦИОННАЯ СТРАТЕГИЯ: ХАРАКТЕРИСТИКА

- представляет собой многократный проход этапов разработки
- основана на частичном определении требований к разрабатываемому программному средству в начале процесса разработки
- требования постепенно уточняются в последовательных циклах разработки
- результат каждого цикла разработки представляет собой очередную поставляемую версию программного средства.

ЭВОЛЮЦИОННАЯ СТРАТЕГИЯ: ХАРАКТЕРИСТИКА



ЭВОЛЮЦИОННАЯ СТРАТЕГИЯ: ДОСТОИНСТВА

1. Возможность уточнения и внесения новых требований в процессе разработки
2. Пригодность промежуточного продукта для использования
3. Возможность управления рисками
4. Обеспечение широкого участия пользователя в проекте, начиная с ранних этапов, что минимизирует возможность разногласий между заказчиками и разработчиками и обеспечивает создание продукта высокого качества.

ЭВОЛЮЦИОННАЯ СТРАТЕГИЯ: НЕДОСТАТКИ

- 1) Неизвестность точного количества необходимых итераций и сложность определения критериев для продолжения процесса разработки на следующей итерации
- 2) Сложность планирования и управления проектом
- 3) Необходимость активного участия пользователей в проекте, что реально не всегда осуществимо
- 4) Необходимость в мощных инструментальных средствах и методах прототипирования
- 5) Возможность сдвига решения трудных проблем на последующие циклы, что может привести к несоответствию полученных продуктов требованиям заказчиков.

ЭВОЛЮЦИОННАЯ СТРАТЕГИЯ: ОБЛАСТЬ ПРИМЕНЕНИЯ

1. Проекты, для которых требования слишком сложны, неизвестны заранее, непостоянны или требуют уточнения
2. Сложные проекты, в том числе:
 - большие долгосрочные проекты
 - проекты по созданию новых, не имеющих аналогов ПС или систем
 - проекты со средней и высокой степенью рисков
 - проекты, для которых нужна проверка концепции, демонстрация технической осуществимости или промежуточных продуктов
3. Проекты, использующие новые технологии.

3. МОДЕЛИ ЖЦ ПО: КАСКАДНАЯ СТРАТЕГИЯ

- Классическая каскадная модель
- Каскадная модель с обратными связями
- V-образная модель

КЛАССИЧЕСКАЯ КАСКАДНАЯ МОДЕЛЬ ЖИЗНЕННОГО ЦИКЛА ПО

1. Подготовка
процесса разработки

I. Выбор модели ЖЦ ПС, методов и
средств разработки

2. Анализ
требований

Сбор требований к ПС, их
систематизация, документирование,
анализ, а также выявление и разрешение
противоречий.

Software Requirement
Specification, SRS

3. Проектирование
программной
архитектуры

Определение архитектуры системы,
внешних условий функционирования,
интерфейсов

КЛАССИЧЕСКАЯ КАСКАДНАЯ МОДЕЛЬ ЖИЗНЕННОГО ЦИКЛА ПО

4. Техническое проектирование ПС

Детальное проектирование программного средства (технический проект компонентов программного объекта).

5. Кодирование и тестирование

Определяется критерий верификации для всех модулей относительно требований, разработка программных модулей, тестирование

6. Сборка и квалификационные испытания ПС

Программный продукт удовлетворяет установленным требованиям.

КАСКАДНАЯ МОДЕЛЬ С ОБРАТНЫМИ СВЯЗЯМИ

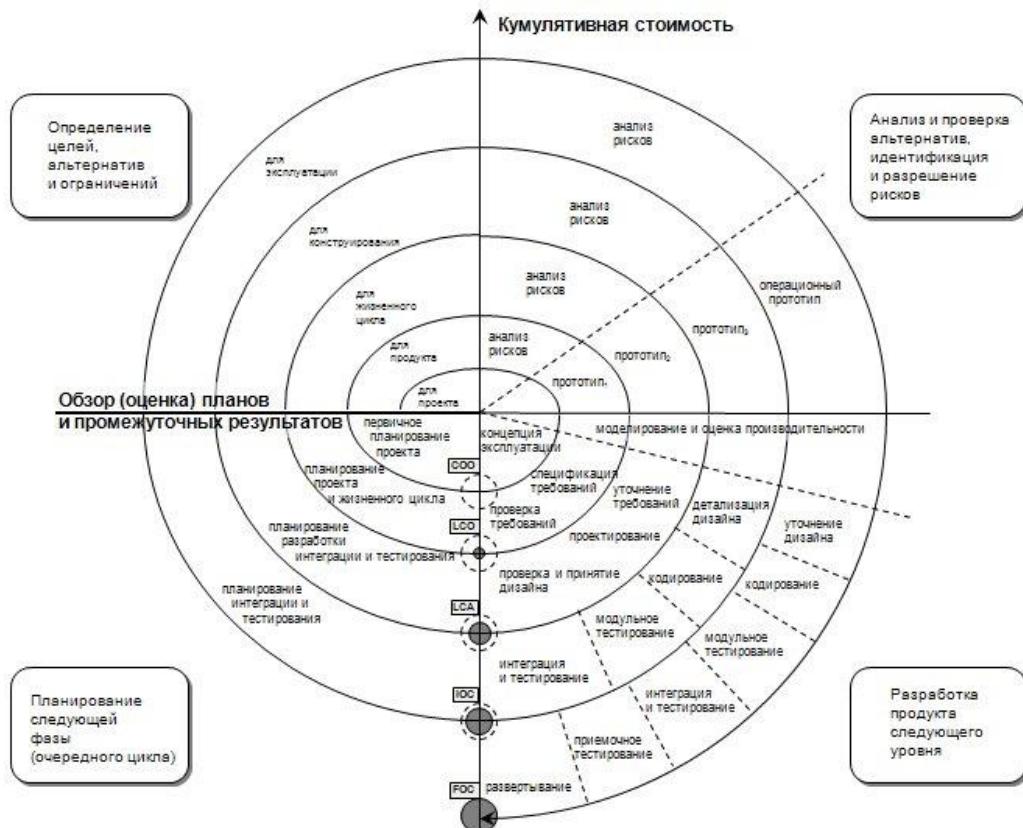
- Организация обратных связей между любыми шагами каскадной модели → возможность исправления продуктов предыдущих шагов процесса разработки
- Сложности планирования и финансирования проекта, достаточно высокий риск нарушения графика разработки по сравнению с классической каскадной моделью

V-ОБРАЗНАЯ МОДЕЛЬ

- Связи между деятельностью по разработке планов испытаний и тестирования и деятельностью по подтверждению результатов соответствующих этапов.



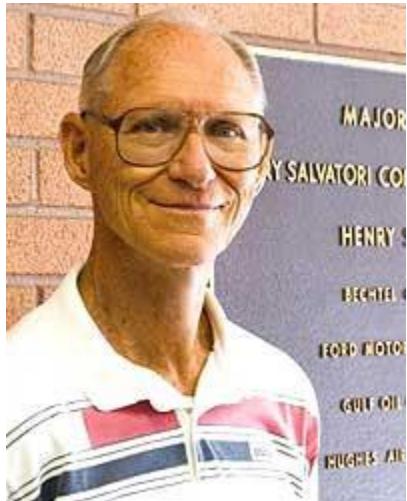
4. СПИРАЛЬНАЯ МОДЕЛЬ (Б.БОЭМ, 1988) ЭВОЛЮЦИОННАЯ СТРАТЕГИЯ



- Отличительная особенность - специальное внимание **рискам**, влияющим на организацию жизненного цикла.

Оригинальная спиральная модель
жизненного цикла разработки по Боэму

СПИРАЛЬНАЯ МОДЕЛЬ (Б.БОЭМ, 1988) ЭВОЛЮЦИОННАЯ СТРАТЕГИЯ



Барри Боэм

Главное достижение спиральной модели состоит в том, что она предлагает спектр возможностей адаптации удачных аспектов существующих моделей процессов жизненного цикла. В то же время, ориентированный на риски подход позволяет избежать многих сложностей, присущих в этих моделях. В определенных ситуациях спиральная модель становится эквивалентной одной из существующих моделей. В других случаях она обеспечивает возможность наилучшего соединения существующих подходов в контексте данного проекта.

СПИРАЛЬНАЯ МОДЕЛЬ (Б.БОЭМ, 1988)

ЭВОЛЮЦИОННАЯ СТРАТЕГИЯ

- Дефицит специалистов.
- Нереалистичные сроки и бюджет.
- Реализация несоответствующей функциональности.
- Разработка неправильного пользовательского интерфейса.
- “Золотая сервировка”, перфекционизм, ненужная оптимизация и оттачивание деталей.
- Непрекращающийся поток изменений.
- Нехватка информации о внешних компонентах, определяющих окружение системы или вовлеченных в интеграцию.
- Недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами.
- Недостаточная производительность получаемой системы.
- “Разрыв” в квалификации специалистов разных областей знаний.

СПИРАЛЬНАЯ МОДЕЛЬ (Б.БОЭМ, 1988)

ЭВОЛЮЦИОННАЯ СТРАТЕГИЯ

- **Квадрант I** – анализ целей, альтернативных вариантов и ограничений – определяются рабочие характеристики, выполняемые функции, стабильность (возможность внесения изменений), аппаратно-программный интерфейс продукта разработки данной фазы или цикла.
- **Квадрант II** – оценка альтернативных вариантов, идентификация и разрешение рисков. Выполняется прототипирование как основа для работ следующего квадранта.
- **Квадрант III** – разработка продукта текущего уровня (разработка и тестирование исходных текстов программ, сборка, тестирование и квалификационные испытания продукта или системы и т.п.)
- **Квадрант IV** – планирование следующей фазы – решение о переходе на цикл следующей фазы разработки или выполнении еще одного цикла текущей фазы разработки

СПИРАЛЬНАЯ МОДЕЛЬ (Б.БОЭМ, 1988)

ЭВОЛЮЦИОННАЯ СТРАТЕГИЯ

- **A. Фаза разработки концепции** (соответствует первому витку спирали)
- **B. Фаза анализа требований** (соответствует второму витку спирали)
- **C. Фаза проектирования системы/программного продукта** (соответствует третьему витку спирали)
- **D. Фаза реализации** (технического проектирования, программирования и сборки)
- **E. Фаза сопровождения и расширения функциональных возможностей** (соответствует пятому витку спирали).

СПИРАЛЬНАЯ МОДЕЛЬ (Б.БОЭМ, 1988) ЭВОЛЮЦИОННАЯ СТРАТЕГИЯ

Движение по спирали:

- каждый виток спирали (итерация) – совокупность стадий каскадной модели
- каждый виток спирали соответствует созданию фрагмента или версии ПО
- с каждой итерацией по спирали строятся все более полные версии ПС.

СПИРАЛЬНАЯ МОДЕЛЬ (Б.БОЭМ, 1988)

ПРЕИМУЩЕСТВА

- 1) Упрощение изменений в проекта при изменении требований заказчика.
- 2) Уменьшение уровня рисков.
- 3) Гибкость в управлении проектом.
- 4) Итерационный подход упрощает повторное использование компонентов.
- 5) Спиральная модель позволяет получить более надежную и устойчивую систему, т.к. ошибки и слабые места обнаружаются и исправляются на каждой итерации

СПИРАЛЬНАЯ МОДЕЛЬ (Б.БОЭМ, 1988)

НЕДОСТАТКИ

- 1) Сложность анализа и оценки рисков при выборе вариантов → требует искусного управления
- 2) Сложность поддержания версий продукта (хранение версий, возврат к ранним версиям, комбинация версий) → необходима поддержка целостности документации;
- 3) Сложность в определении момента перехода на следующий этап → временные ограничения на каждый из этапов жизненного цикла.

СПИРАЛЬНАЯ МОДЕЛЬ (Б.БОЭМ, 1988)

ОБЛАСТЬ ПРИМЕНЕНИЯ

- Когда пользователи не уверены в своих потребностях или когда требования слишком сложны и могут меняться в процессе выполнения проекта и необходимо прототипирование для анализа и оценки требований
- Когда достижение успеха не гарантировано и необходима оценка рисков продолжения проекта
- Когда проект является сложным, дорогостоящим и обоснование его финансирования возможно только в процессе его выполнения
- Когда речь идет о применении новых технологий, что связано с риском их освоения и достижения ожидаемого результата
- При выполнении очень больших проектов, которые в силу ограниченности ресурсов можно делать только по частям.

УПРОЩЕННЫЕ ВАРИАНТЫ СПИРАЛЬНОЙ МОДЕЛИ

- Модель Института качества SQI
- Модель Института Управления проектами PMI
- Модель «win-win»
- Спиральная модель Консорциума по вопросам разработки программного обеспечения

ВЫБОР МОДЕЛИ ЖЦ ПО

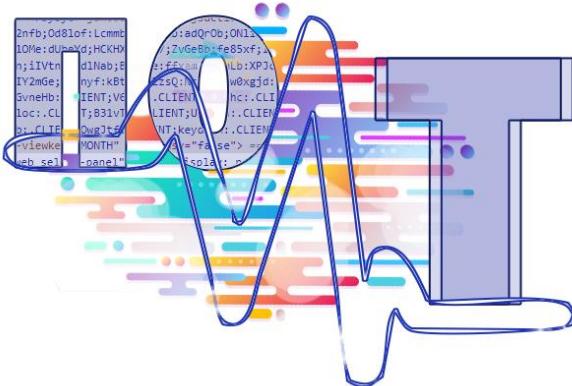
Критерии категории требований	Каскадная	V-модель	Инкрементная	Эволюционная
Являются ли требования к проекту легко определимыми и реализуемыми?	Да	Да	Нет	Нет
Могут ли требования быть сформулированы в начале ЖЦ?	Да	Да	Да	Нет
Часто ли будут изменяться требования на протяжении ЖЦ?	Нет	Нет	Нет	Да
Нужно ли реализовать основные требования на ранних этапах разработки?	Нет	Нет	Да	Да
•	•	•	•	•

ВЫБОР МОДЕЛИ ЖЦ ПО

Критерии категории команды разработчиков проекта	Каскадная	V-модель	Инкрементная	Эволюционная
Являются ли проблемы предметной области проекта новыми для большинства разработчиков?	Нет	Нет	Нет	Да
Изменяются ли роли участников проекта на протяжении ЖЦ?	Нет	Нет	Да	Да
Является ли структура процесса разработки более значимой для разработчиков, чем гибкость?	Да	Да	Да	Нет

Методологии разработки программного обеспечения

Лекция 3



План лекции

1. Классификация процессов разработки. Быстрая разработка приложений RAD
2. Рациональный унифицированный процесс RUP
3. Agile Manifesto. Гибкие методологии

1. Классификация

В зависимости от условий разработки выделяют :

Прогностические («тяжеловесные») процессы

Предполагают планирование всего объема работ, достаточно большой объем документации

Основная цель – отделить успешные практики разработки и сопровождения ПО от конкретных людей, умеющих их применять

Многочисленные вспомогательные действия имеют целью выполнение успешной разработки с помощью имеющихся работников, не обязательно являющихся суперпрофессионалами

1. Классификация

Адаптивные или облегченные, гибкие, «живые» (agile) процессы разработки

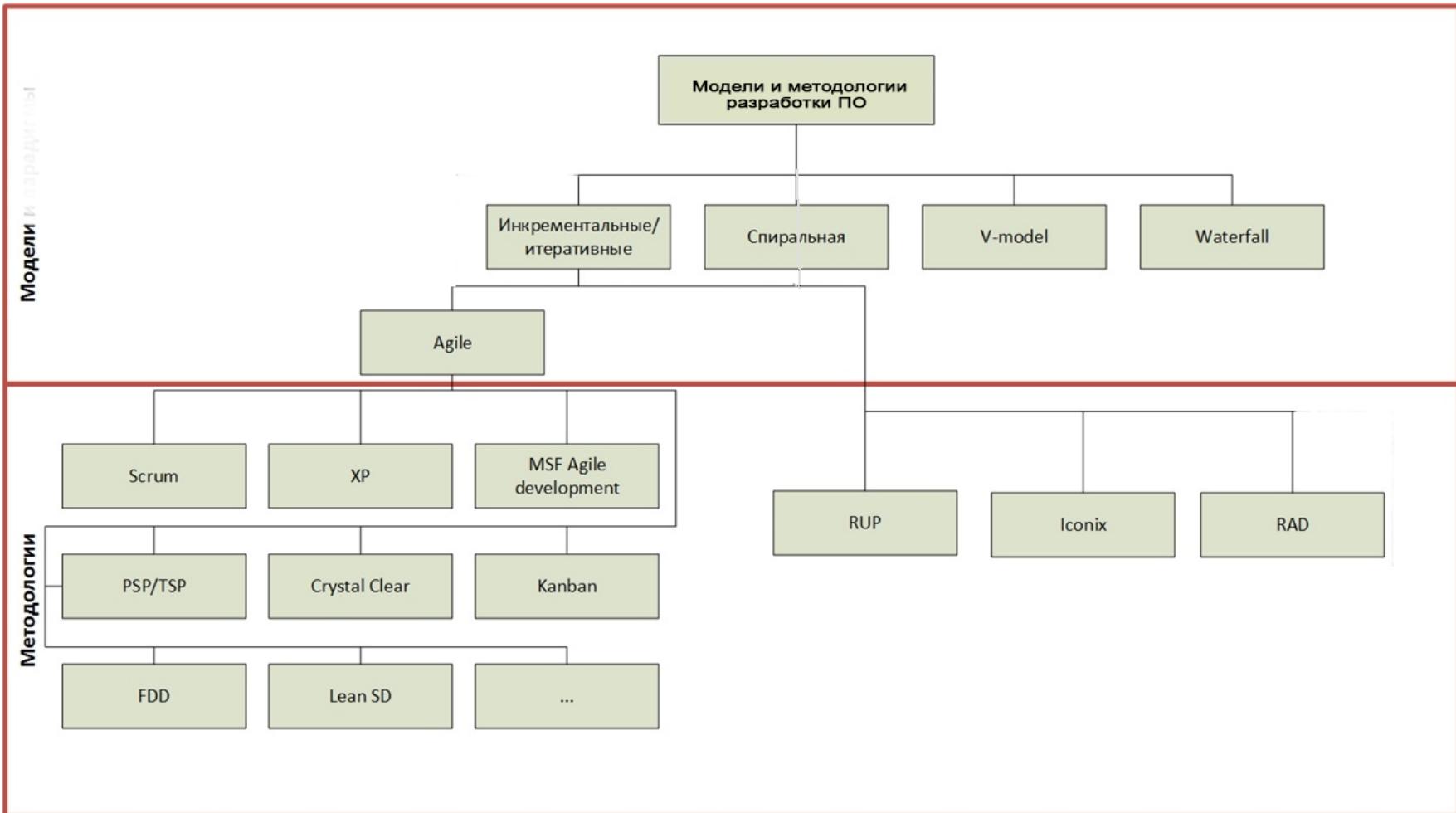
Не требуют столь жесткой регламентации, допускают возможность частых и существенных изменений требований заказчиков

Делают упор на использование хороших разработчиков, а не хорошо отлаженных процессов разработки

Избегают фиксации четких схем действий, чтобы обеспечить большую гибкость в каждом конкретном проекте и не требуют создания дополнительных промежуточных документов

МЕТОДОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Классификация

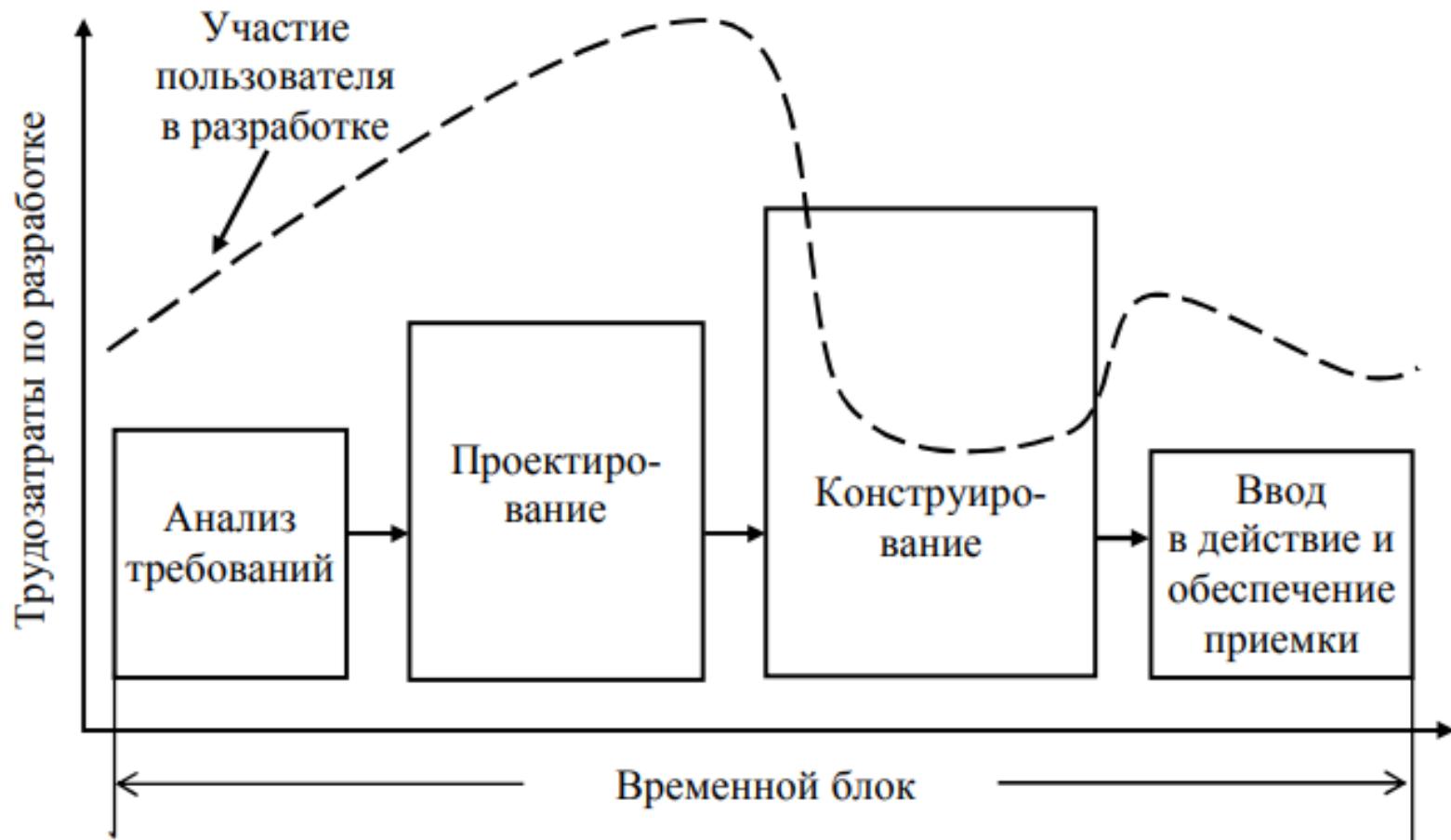


1. Быстрая разработка приложений : характеристика

- небольшая команда разработчиков (до 10 человек);
- короткий, но тщательно проработанный производственный график (от 2 до 6 месяцев);
- повторяющийся цикл
- эволюционная или инкрементная стратегии разработки
- мощные инструментальные средства разработки – визуальные среды проектирования и программирования
- прототипирование

RAD (Rapid Application Development)

1. Базовая RAD-модель



RAD (Rapid Application Development)

1. RAD, основанная на моделировании

- 1) Функциональное или бизнес-моделирование. Моделируется информационный поток между бизнес-функциями;
- 2) Моделирование данных или информационное моделирование. Информационный поток → набор объектов данных (свойства, атрибуты) → отношения между объектами;
- 3) Моделирование обработки (поведения). Определяются преобразования объектов данных, обеспечивающие реализацию бизнес функций;
- 4) Автоматическая кодогенерация;
- 5) Тестирование и объединение.

Функциональное
моделирование

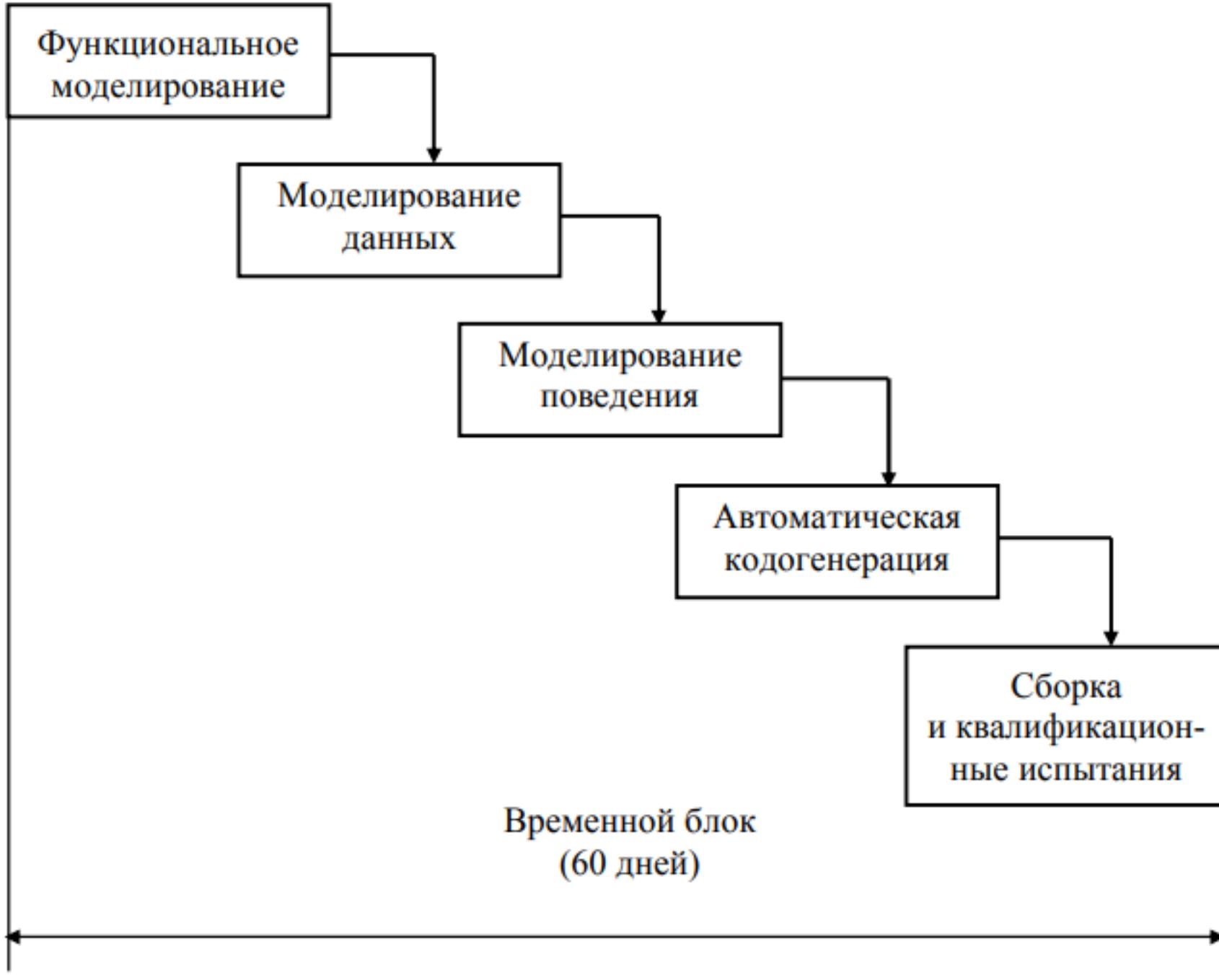
Моделирование
данных

Моделирование
поведения

Автоматическая
кодогенерация

Сборка
и квалификацион-
ные испытания

Временной блок
(60 дней)



RAD (Rapid Application Development)

1. RAD: достоинства

- 1) Сокращение продолжительности цикла разработки и всего проекта в целом, сокращение количества разработчиков, а следовательно, и стоимости проекта за счет использования мощных инструментальных средств;
- 2) сокращение риска, связанного с выполнением графика, за счет использования принципа временного блока и связанное с этим упрощение планирования;
- 3) сокращение риска, связанного с неудовлетворенностью заказчика разработанным продуктом, за счет его привлечения на постоянной основе к циклу разработки;
- 4) возможность повторного использования существующих компонентов.

1. RAD: недостатки

- 1) Необходимость в высококвалифицированных разработчиках, умеющих работать с инструментальными средствами разработки;
- 2) жесткость временных ограничений на разработку прототипа;
- 3) сложность ограничения затрат и определения сроков завершения работы над проектом
- 4) неприменимость в условиях высоких технических рисков, при использовании новых технологий.

1. RAD-модель: область применения

- 1) При разработке систем и продуктов, для которых характерно хотя бы одно из следующих свойств:
 - поддаются моделированию
 - имеют небольшой размер
 - относятся к известной разработчикам предметной области
 - являются информационными системами
 - требования для них хорошо известны
 - имеются компоненты пригодные к повторному использованию;

RAD-модель: область применения

- 2) если пользователь может принимать постоянное участие в процессе разработки;
- 3) если в проекте заняты разработчики, обладающие достаточными навыками в использовании инструментальных средств разработки;
- 4) при выполнении проектов в сокращенные сроки (как правило, не более чем за 60 дней);
- 5) при разработке ПС, для которых требуется быстрое наращивание функциональных возможностей на последовательной основе;
- 6) при невысокой степени технических рисков;
- 7) в составе других моделей жизненного цикла.

RAD (Rapid Application Development)

• Функциональное моделирование



RAD (Rapid Application Development)

- Функциональное моделирование



- Декомпозиция

RAD (Rapid Application Development)

- Функциональное моделирование



2. Rational Unified Process, RUP

- **Цель RUP:** гарантировать высокое качество программного продукта, отвечающего потребностям конечных пользователей, в пределах предсказуемого временного графика и бюджета.
- RUP обеспечивает строгий подход к назначению задач и ответственности в пределах группы разработки

Rational Unified Process, RUP : характеристика

- Корпорация **Rational Software** (подразделение IBM)
 - formalizovala технологический процесс разработки ПО
 - выпустила структурированную базу знаний под названием Rational Unified Process (**RUP**) -
 - методические рекомендации ведущих разработчиков по эффективному созданию приложений и систем.
- Используется язык моделирования Unified Modelling Language (UML).

RUP: характеристика

- Заключается в создании и обслуживании моделей
- Сосредотачивает внимание на первоначальной разработке и компоновке устойчивой архитектуры программы
- Поддерживает объектно-ориентированный подход к разработке ПО с перестраиваемой конфигурацией
- Поддерживается инструментальными средствами, которые автоматизируют большинство действий процесса

Принципы RUP

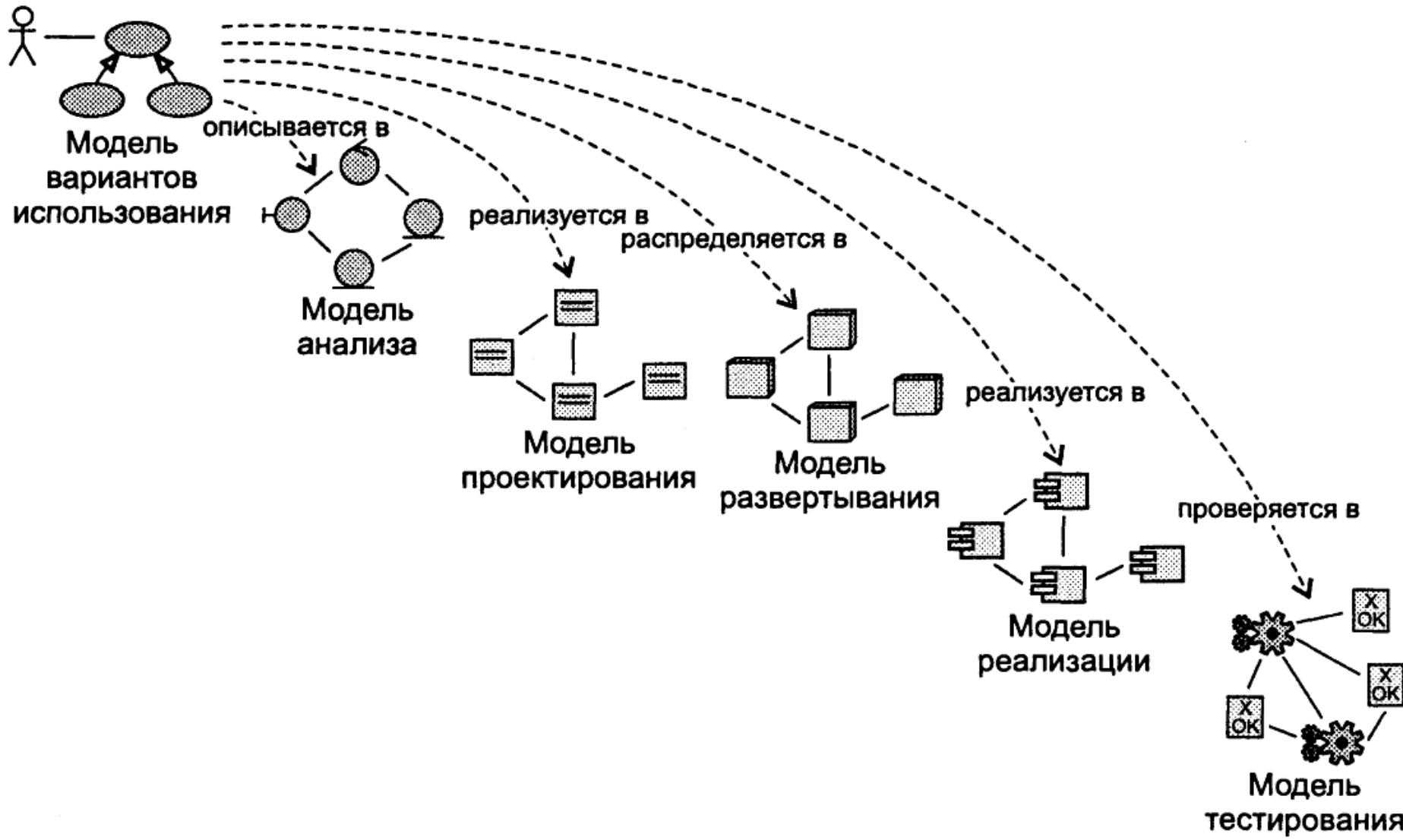
- Ранняя идентификация и непрерывное (до окончания проекта) устранение основных рисков.
- Концентрация на выполнении требований заказчиков к исполняемой программе
- Ожидание изменений в требованиях, проектных решениях и реализации в процессе разработки
- Архитектура, реализуемая и тестируемая на ранних стадиях проекта
- Постоянное обеспечение качества на всех этапах разработки
- Работа над проектом в сплочённой команде, ключевая роль в которой принадлежит архитекторам

Статический аспект RUP

Представлен четырьмя основными элементами:

- артефакты (рабочие продукты)
 - это некоторые продукты проекта, порождаемые или используемые в нем при работе над окончательным продуктом:
 - модель, элемент модели,
 - документ,
 - исходный код, план

Модели унифицированного процесса



Статический аспект RUP

- «**Роль**» (role) определяет
 - поведение
 - ответственность личности или группы личностей, составляющих проектную команду.

Одна личность может играть в проекте много различных ролей.

- Под **видом деятельности** конкретного исполнителя понимается единица выполняемой им работы (соответствует понятию технологической операции).

Каждый вид деятельности связан с конкретной ролью.

Дисциплина (discipline)

Дисциплина (discipline) соответствует понятию технологического процесса и представляет собой последовательность действий, приводящую к получения значимого результата.

Основные дисциплины:

- построение бизнес - моделей
- определение требований
- анализ и проектирование
- реализация, кодирование
- тестирование
- развертывание, внедрение

Вспомогательные:

- управление конфигурацией и изменениями
- управление проектом
- создание инфраструктуры.

РАЦИОНАЛЬНЫЙ УНИФИЦИРОВАННЫЙ ПРОЦЕСС

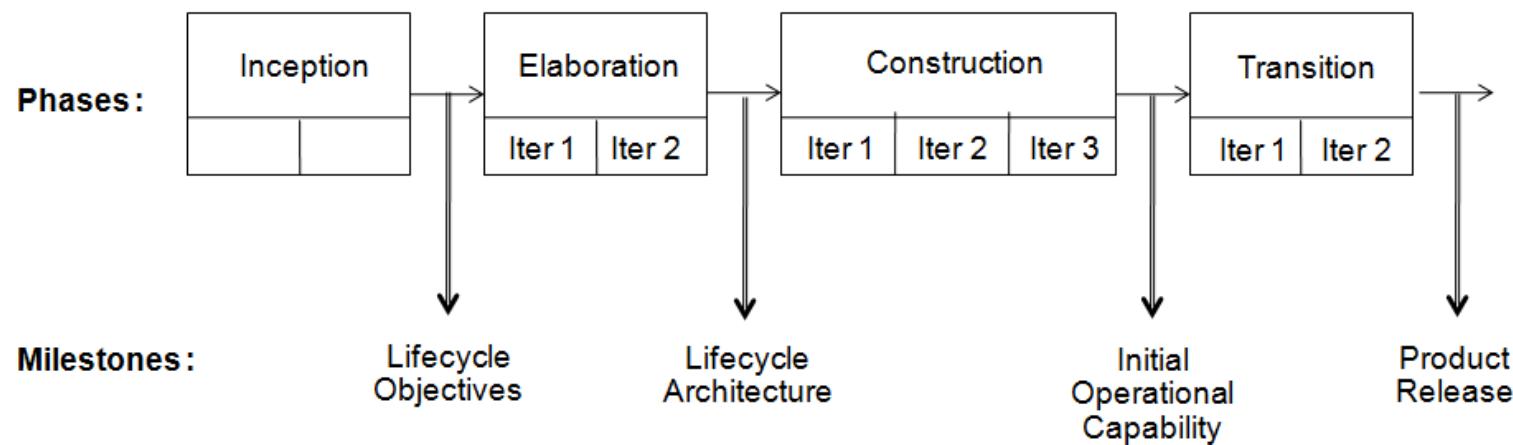
Структура модели

Состоит из четырех фаз, каждая из которых включает в себя одну или несколько итераций:

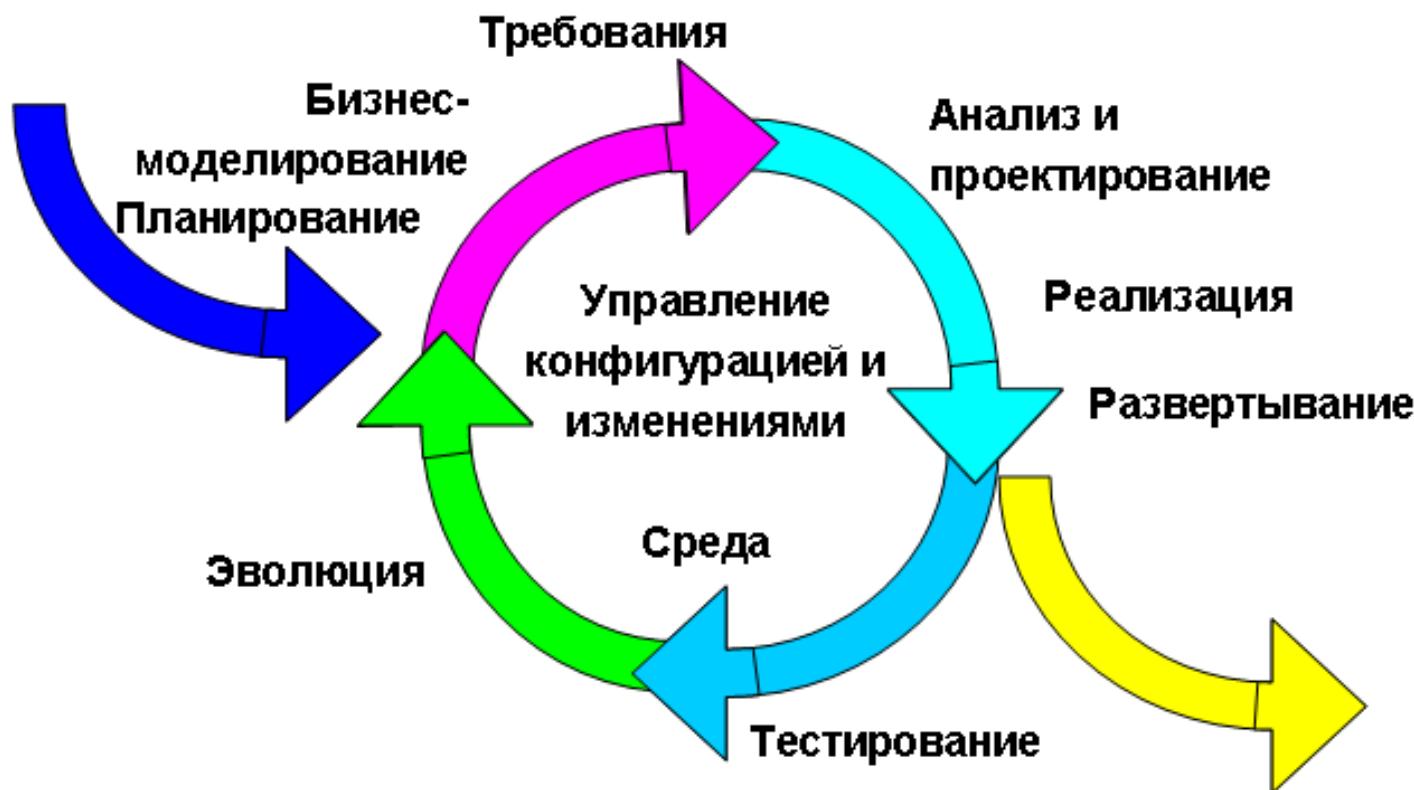
1. Начало (Inception)
2. Проектирование (Elaboration)
3. Построение (Construction)
4. Внедрение (Transition)

У каждой фазы есть

1. Цель,
2. Основная деятельность
3. Контрольная точка.



Итерационный цикл

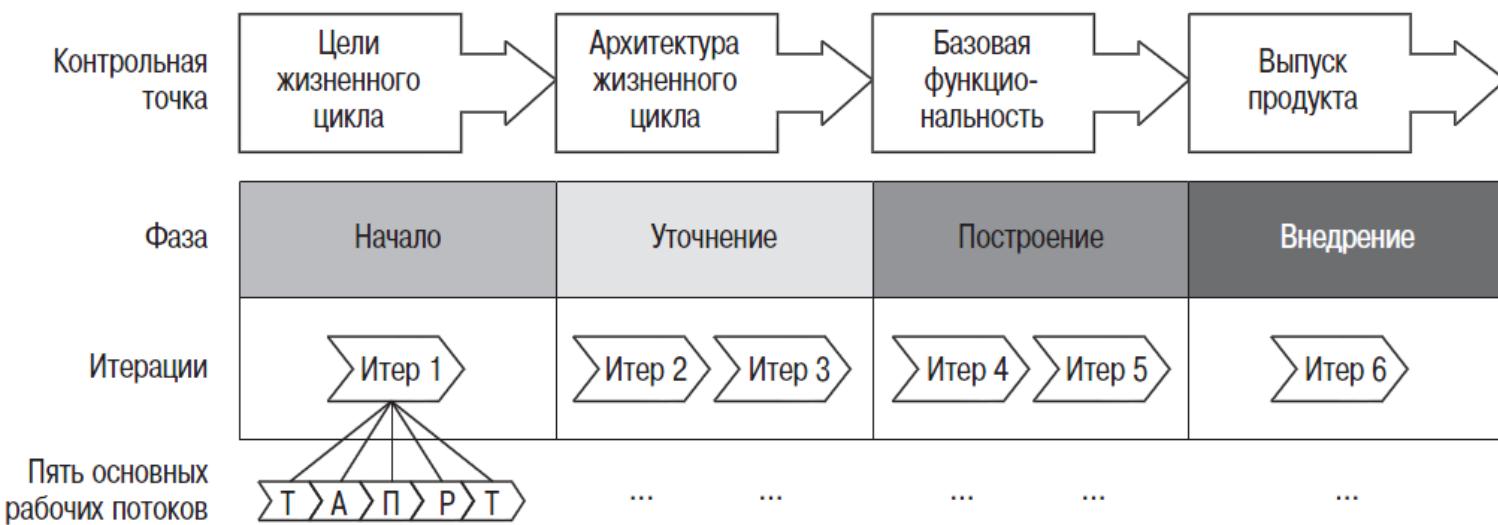


Итерационный цикл

- Итерационный цикл основывается на
 - постоянном расширении
 - дополнении системы в процессе нескольких итераций с периодической обратной связью
 - адаптацией добавляемых модулей к существующему ядру системы.
- Система постоянно разрастается шаг за шагом, поэтому такой подход называют итерационным и инкрементным.
- Такой подход исключает
 - слишком быстрое написание кода (без детальной проработки)
 - чрезмерно длительный этап детального проектирования
 - построения моделей без обратной связи.

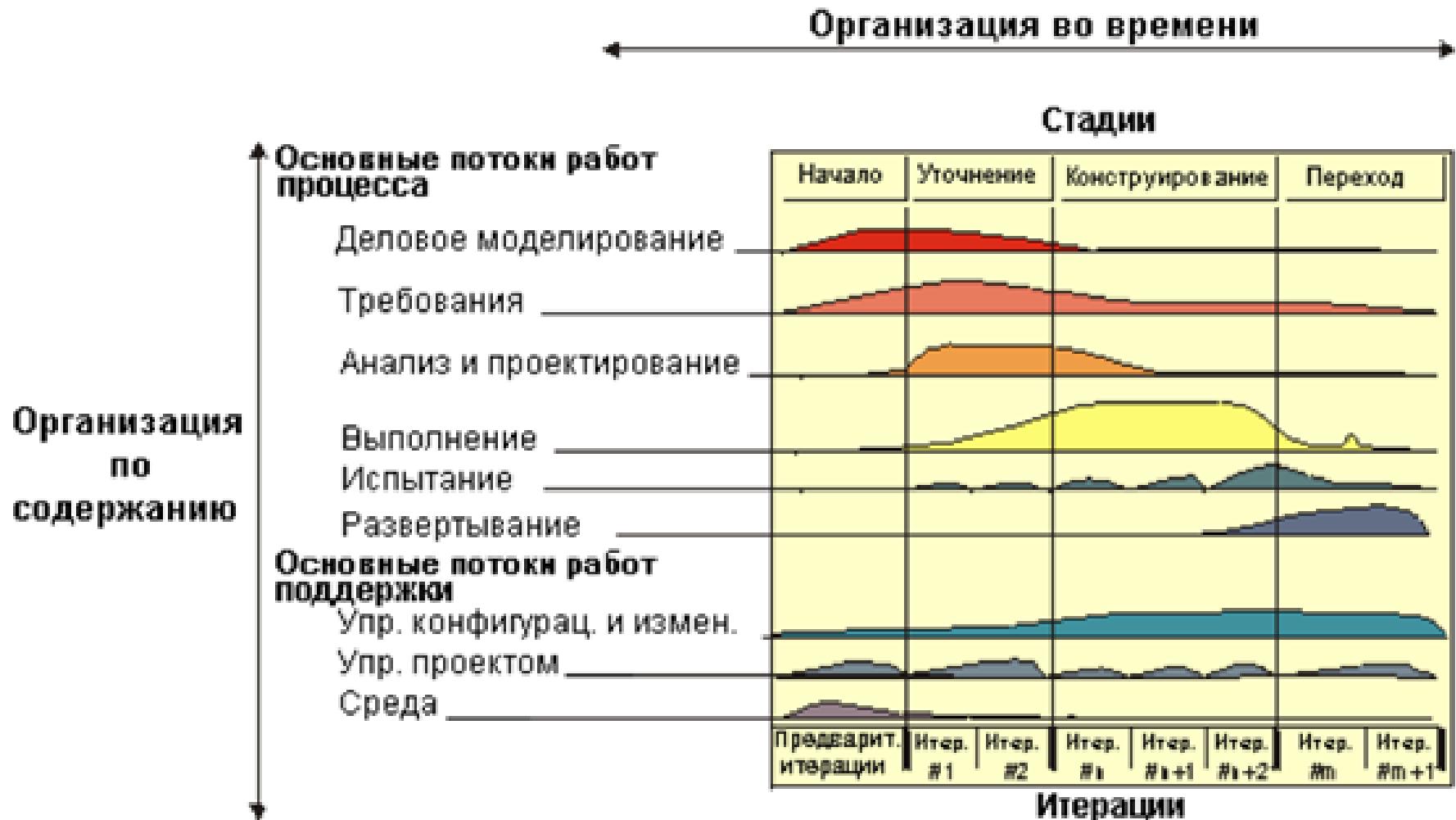
Итерационный цикл

- Каждая итерация включает в себя некоторые или большую часть дисциплин разработки: выявление требований, анализ, проектирование, реализация, тестирование.
- Каждая последующая итерация строится на результатах предыдущих, развивает и усовершенствует систему до тех пор, пока не будет создан конечный продукт.
- Более ранние итерации больше концентрируются на требованиях, анализе и проектировании, более поздние - на реализации и тестировании.



РАЦИОНАЛЬНЫЙ УНИФИЦИРОВАННЫЙ ПРОЦЕСС

Общее представление RUP



Общее представление RUP

Общее представление RUP в двух измерениях:

- горизонтальное измерение представляет
 - время
 - отражает динамические аспекты процессов
 - оперирует такими понятиями
 - стадии,
 - итерации
 - контрольные точки;
- вертикальное измерение отражает
 - статические аспекты процессов
 - оперирует такими понятиями
 - виды деятельности (технологические операции),
 - рабочие продукты,
 - исполнители
 - дисциплины (технологические процессы).

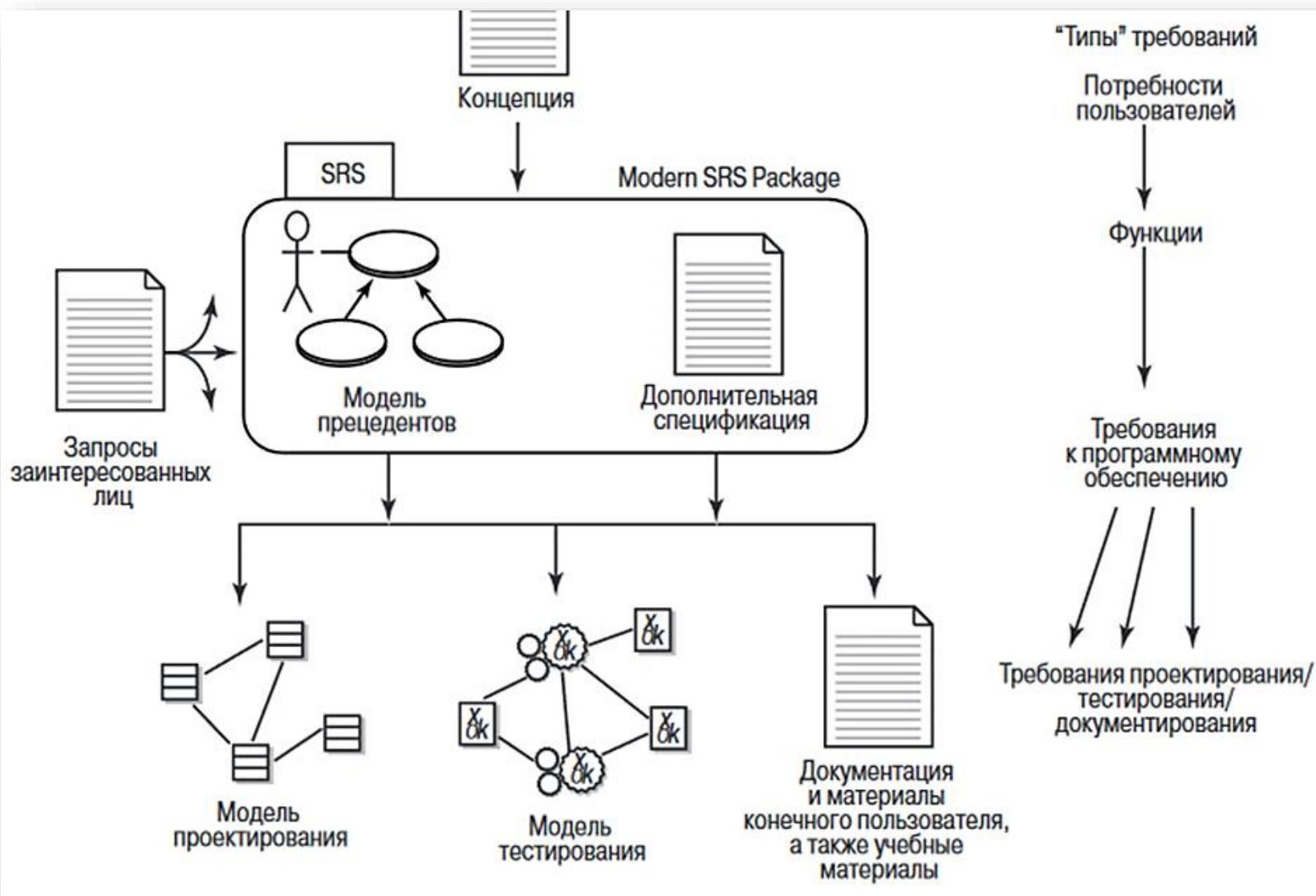
Начало (Inception)

На этом этапе:

- Формируются видение и границы проекта.
- Создается экономическое обоснование (business case).
- Определяются основные требования, ограничения и ключевая функциональность продукта.
- Создается базовая версия модели прецедентов.
- Оцениваются риски.

РАЦИОНАЛЬНЫЙ УНИФИЦИРОВАННЫЙ ПРОЦЕСС

Принципы управления требованиями в RUP



Артефакты рабочего процесса разработки требований

- **Запросы заинтересованных лиц** – коллекция различных запросов:
 - формальные запросы изменений,
 - потребности или другие пожелания заинтересованных лиц на протяжении жизненного цикла проекта, которые могут повлиять на требования к продукту
- **Документ-концепция (Vision document):**
 - кратко характеризует концепцию рассматриваемой системы: основные характеристики, функции, потребности заинтересованных лиц, а также основные предоставляемые услуги.

Артефакты рабочего процесса разработки требований

- **Модель прецедентов**, которая представляет собой организованный набор прецедентов, составляющих основную массу требований.
- **Дополнительные спецификации**, фиксирующие все требования, которые невозможно непосредственно связать с конкретными прецедентами:
 - нефункциональные требования и ограничения проектирования.
- Эти артефакты вместе образуют единую форму – пакет **Modern SRS Package**.

Modern Software Requirements Specification – Спецификация требований к программному обеспечению

- Представляет собой набор артефактов, полностью описывающих внешнее поведение системы.
- Создает концептуальную модель создаваемой системы.
- Исходная информация для создания Modern SRS Package – документ-концепция (Vision document), определяющий потребности пользователей, цели, задачи, целевые рынки и функции системы
- В пакете Modern SRS Package основное внимание уделяется деталям реализации этих функций.

Уточнение (Elaboration)

На этапе **Уточнения (фаза развития)** производится анализ предметной области и построение исполняемой архитектуры.

Фаза развития — это первая последовательность итераций, в течение которых решаются следующие задачи:

- Изучается и стабилизируется большая часть требований
- Обосновываются и устраняются основные риски
- Реализуются и тестируются базовые архитектурные элементы

Уточнение (Elaboration)

На начальных итерациях разрабатывается и обосновывается базовая архитектура:

- Разработка и реализация системы “не вглубь, а вширь”
 - означает идентификацию отдельных процессов, слоев, пакетов и подсистем, их высокоуровневых функций и интерфейсов, модули могут содержать в основном заглушки.
- Уточнение локальных и удаленных интерфейсов между модулями (включая параметры и возвращаемые значения).
- Интегрирование существующих компонентов
- Реализация упрощенных сценариев, обеспечивающих параллельное проектирование, программирование и тестирование основных компонентов

Планирование следующей итерации

- Требования и итерации систематизируются в соответствии с рисками, границами и критичностью
 - эти критерии используются для распределения работы по итерациям.
- **Риск** (risk) — это техническая сложность или другой фактор, например, отсутствие информации о необходимых затратах или ресурсах.
- **Границы** (coverage) — на начальных итерациях нужно определить все основные части системы, т.е. выполнить реализацию множества компонентов “не вглубь, а вширь”.
- **Критичность** (criticality) — требуется реализовать функции, имеющие важное значение для системы

Планирование следующей итерации

- Прецеденты или их отдельные сценарии ранжируются с целью определения приоритетов при реализации
- На начальных итерациях реализуются прецеденты с высоким рейтингом
- Ранжирование выполняется перед началом первой итерации, затем перед началом второй и т.д.
 - Такой план является адаптивным, а не зафиксированным на начальной стадии проекта

Приоритет	Требование (прецедент или свойство)	Комментарий
Высокий	Оформление продажи	Самый высокий приоритет
	Регистрация	Сложно добавить позднее
Средний	Поддержка пользователей	Влияет на безопасность

Уточнение (Elaboration)

Анализ предметной области и построение исполняемой архитектуры включает в себя:

- Документирование требований (включая детальное описание для большинства прецедентов использования).
- Спроектированную, реализованную и оттестированную исполняемую архитектуру.
- Обновленное экономическое обоснование и более точные оценки сроков и стоимости.
- Сниженные основные риски.

Успешное выполнение фазы означает достижение вехи архитектуры жизненного цикла (Lifecycle Architecture Milestone).

Построение (Construction)

Во время этой фазы происходит реализация большей части функциональности продукта.

Фаза **Построение** завершается

- первым внешним релизом системы
- руководством пользователя
- описанием текущей реализации
- вехой начальной функциональной готовности (Initial Operational Capability).

Внедрение (Transition)

Во время фазы **Внедрение** создается финальная версия продукта и передается от разработчика к заказчику.

Это включает в себя

- программу бета-тестирования,
- обучение пользователей,
- определение качества продукта.

В случае, если качество не соответствует ожиданиям пользователей или критериям, установленным в фазе Начало, фаза Внедрение повторяется снова.

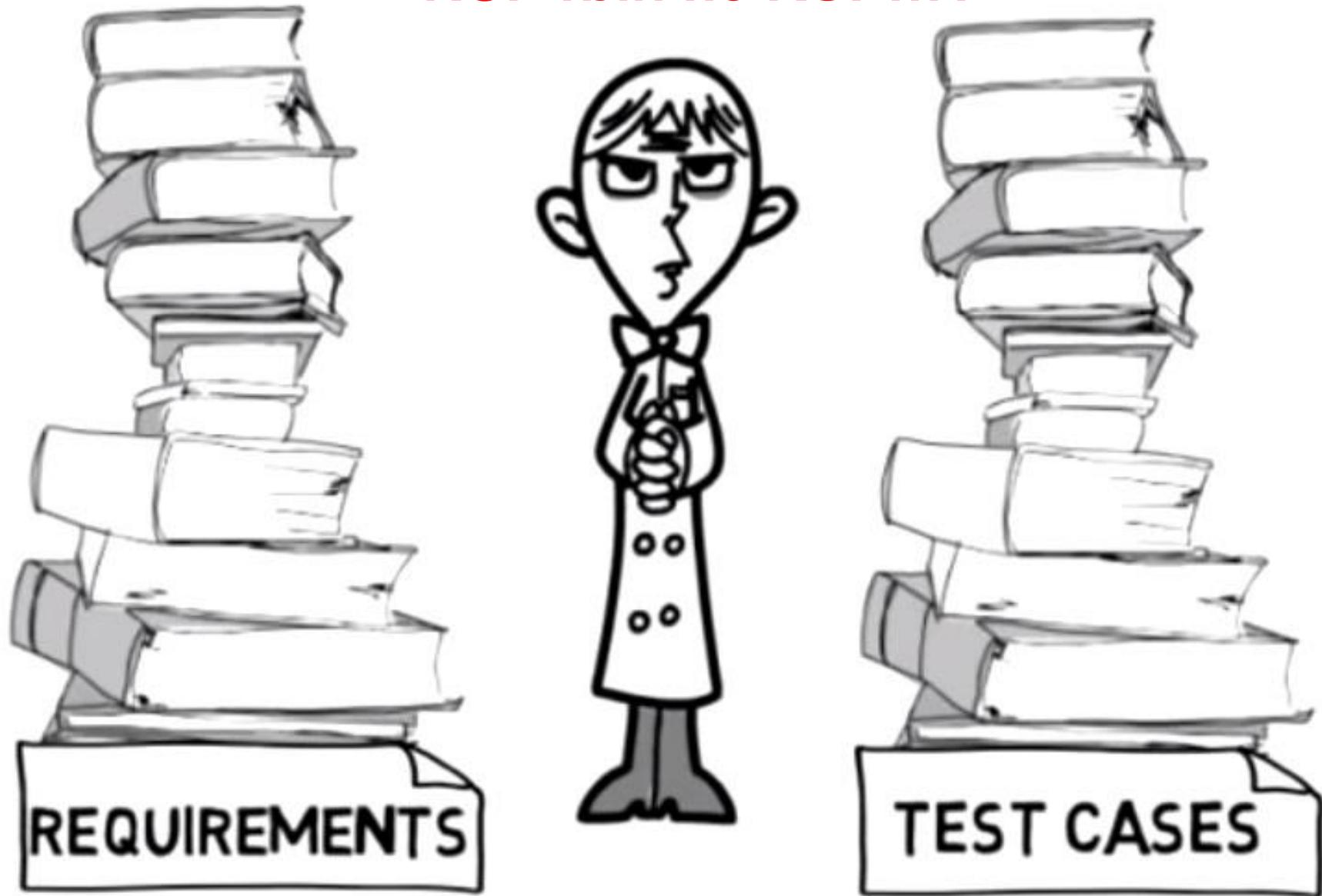
Выполнение всех целей означает достижение вехи готового продукта (Product Release) и завершение полного цикла разработки.

РАЦИОНАЛЬНЫЙ УНИФИЦИРОВАННЫЙ ПРОЦЕСС

Степень активности подпроцессов на разных этапах RUP

	Начало	Развитие	Построение	Внедрение
Требования	Высокая	Высокая	Низкая	Нет
Анализ и проектирование	Низкая	Высокая	Средняя	Нет
Реализация	Нет	Низкая	Высокая	Низкая
Тестирование	Нет	Низкая	Высокая	Средняя
Развертывание	Нет	Нет	Средняя	Высокая
Управление проектом	Средняя	Средняя	Средняя	Средняя
Управление конфигурированием	Низкая	Низкая	Высокая	Высокая

RUP или не RUP...?

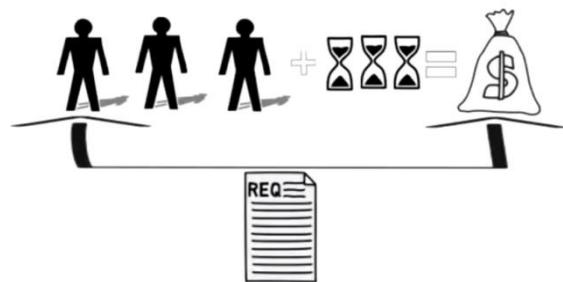


Когда нам нужен RUP?

- Команда разделена
- Необходимость соблюдения формальных процессов общения
- Сложные проекты
- Тестирование основано на требованиях
- Требуются точные оценки



TEST SCENARIO
ACCEPTANCE TEST



Agile: история

- В феврале 2001 года на лыжном курорте The Lodge at Snowbird в горах Юты несколько известных разработчиков ПО (Kent Beck, Martin Fowler, Alistair Cockburn и др.) пришли к соглашению о необходимости документального оформления новых идей в организации процесса разработки
- Ими был согласован и представлен профессиональному сообществу документ под названием *Agile Manifesto*

Текст Agile-манифеста

- «Мы постоянно открываем для себя более совершенные методы разработки программного обеспечения, занимаясь разработкой непосредственно и помогая в этом другим.
- Благодаря проделанной работе мы смогли осознать, что:
 - **Люди и взаимодействие** важнее **процессов и инструментов**
 - **Работающий продукт** важнее **исчерпывающей документации**
 - **Сотрудничество с заказчиком** важнее **согласования условий контракта**
 - **Готовность к изменениям** важнее **следования первоначальному плану**
- То есть, не отрицая важности того, что справа, мы всё-таки больше ценим то, что слева.»

Принципы

- Agile Manifesto декларирует следующие 12 принципов «живой» разработки:
 - наивысшим приоритетом является **удовлетворение заказчика** посредством ранней и постоянной поставки ценного ПО;
 - **изменяющиеся требования** приветствуются, даже на поздних стадиях разработки;

Принципы

- частая поставка рабочего программного обеспечения (каждый месяц или неделю или ещё чаще);
- тесное, ежедневное **общение заказчика с разработчиками** на протяжении всего проекта;
- проектом занимаются **мотивированные личности**, которые обеспечены нужными условиями работы, поддержкой и доверием;

Принципы

- самым эффективным методом передачи информации команде разработчиков и внутри неё является **личное общение**;
- **работающее программное обеспечение** — лучший измеритель прогресса;
- спонсоры, разработчики и пользователи должны все время выдерживать **постоянный темп**;

Принципы

- постоянное внимание улучшению **технического мастерства и удобному дизайну**;
- **простота** — искусство не делать лишней работы;
- лучшие технические требования, дизайн и архитектура получаются у **самоорганизующейся команды**;
- **постоянная адаптация** к изменяющимся обстоятельствам.

Адаптивные методологии

- **Crystal Methods** (1992 г.) – семейство методологий, послужившее отправной точкой в развитии идей адаптивной разработки; наиболее известная методология этого семейства Crystal Clear
- **Agile Unified Process** – упрощенная версия RUP
- **Agile Data Method** – группа итеративных методов разработки программного обеспечения, в которых требования и решения достигаются в рамках сотрудничества разных кросс-функциональных команд

Адаптивные методологии

- **DSDM** (Dynamic Systems Development Method, 1994 г.) – итеративный и инкрементный подход, придающий особое значение продолжительному участию в процессе пользователя/потребителя; основан на концепции быстрой разработки приложений (RAD)
- **Экстремальное программирование** (XP, 1995 г.) – декларирует двенадцать основных приёмов программирования

Адаптивные методологии

- **Scrum** (1995 г.) – методология, делающая основной акцент на общении с заказчиком, на общении внутри команды, на достижении самоорганизации и высокого уровня самосовершенствования

3. XP-процесс

- Экстремальное программирование (eXtreme Programming, XP-процесс) – одна из наиболее популярных адаптивных моделей
- Авторы методологии – Кент Бек, Уорд Каннингем, Мартин Фаулер
- XP-процесс ориентирован на разработку качественного продукта группами малого и среднего размера в условиях неопределенных или быстро меняющихся требований

XP-процесс

- Основная идея XP-процесса – устраниить высокую стоимость внесения изменений. Это достигается путем резкого (до двух недель) сокращения длительности отдельных итераций
- Базовыми действиями являются:
 - кодирование,
 - тестирование,
 - выслушивание заказчика,
 - проектирование

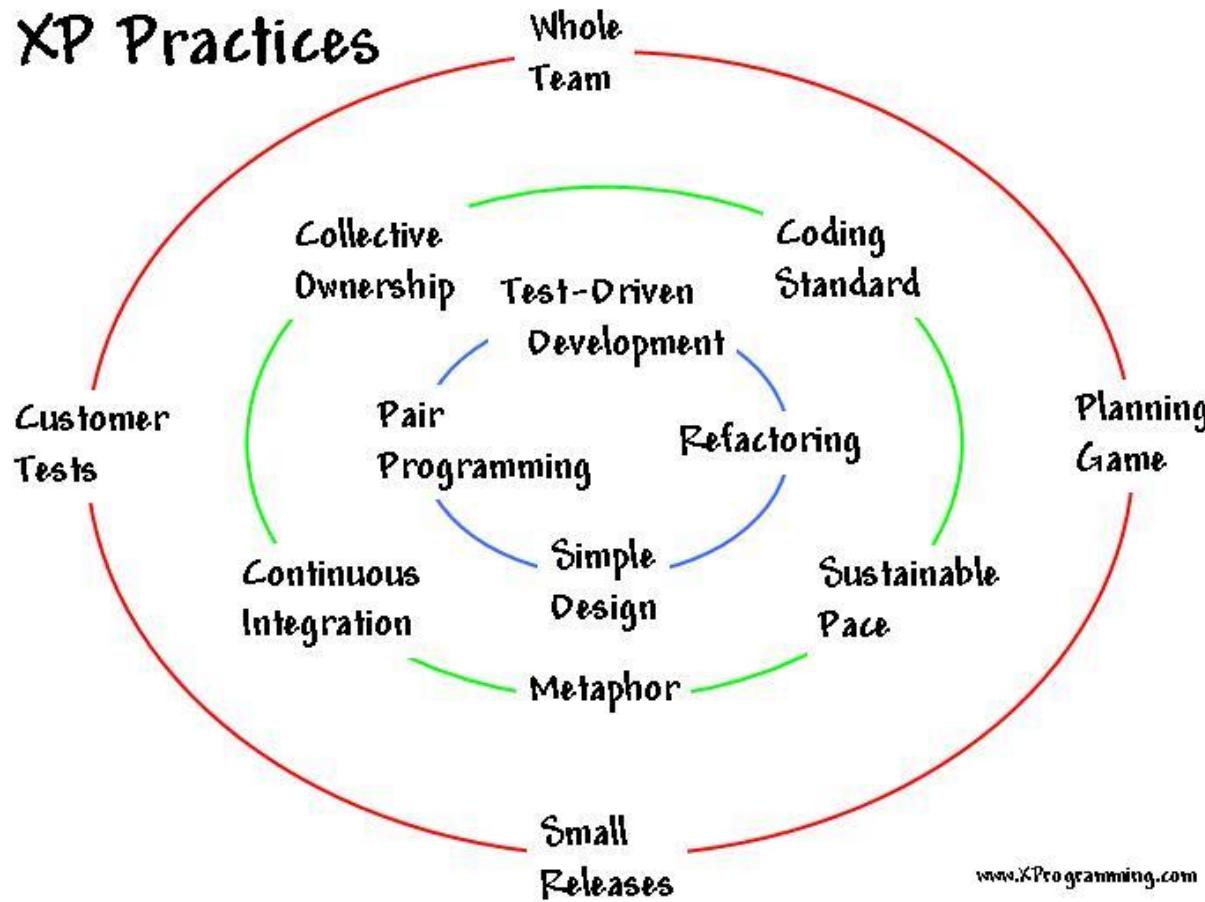
Принципы и практики XP

- Короткий цикл обратной связи (Fine scale feedback)
 - Разработка через тестирование (Test driven development)
 - Игра в планирование (Planning game)
 - Заказчик всегда рядом (Whole team, Onsite customer)
 - Парное программирование (Pair programming)
- Непрерывный, а не пакетный процесс
 - Непрерывная интеграция (Continuous Integration)
 - Рефакторинг (Design Improvement, Refactor)
 - Частые небольшие релизы (Small Releases)

Принципы и практики XP

- Понимание, разделяемое всеми
 - Простота (Simple design)
 - Метафора системы (System metaphor)
 - Коллективное владение кодом (Collective code ownership) или выбранными шаблонами проектирования (Collective patterns ownership)
 - Стандарт кодирования (Coding standard or Coding conventions)
- Социальная защищенность программиста (Programmer welfare):
 - 40-часовая рабочая неделя (Sustainable pace, Forty hour week)

XP в картинках



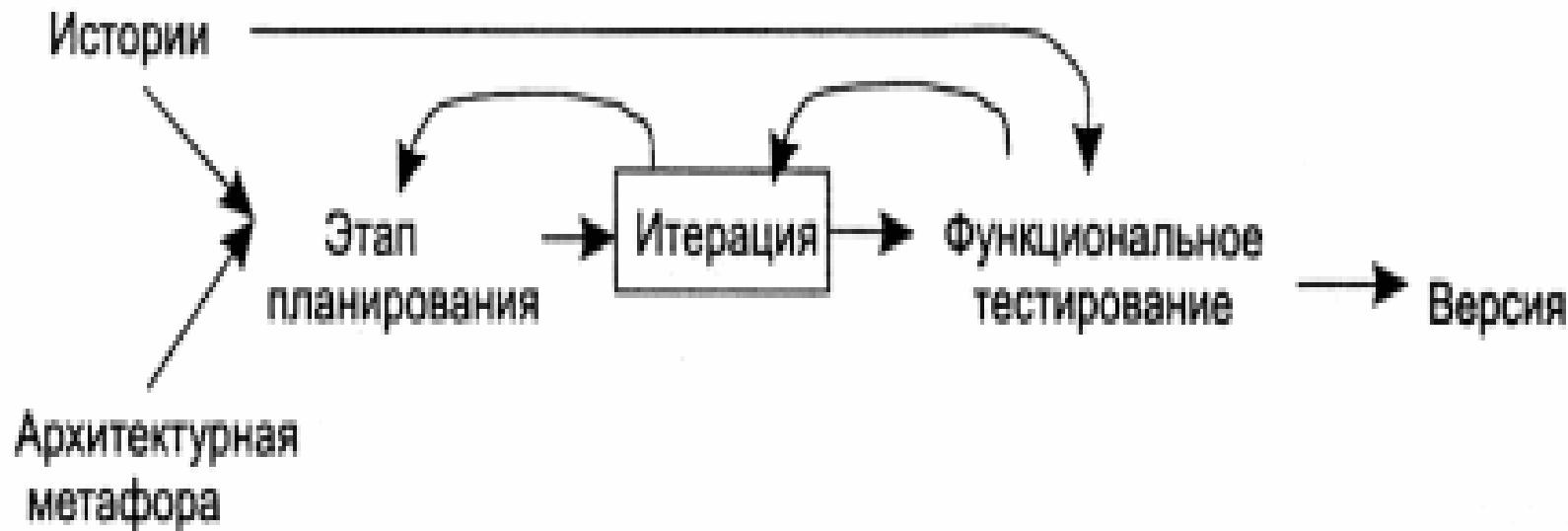
Практики XP

- *Метафора* – вся разработка ведется на основе простой, общедоступной истории о том, как работает система
- *Простое проектирование* – принимаются наиболее простые из возможных проектные решения
- *Непрерывное тестирование* как отдельных модулей, так и системы в целом; входным критерием для написания кода является отказавший тестовый вариант
- *Реорганизация (Refactoring)* – улучшение структуры системы при сохранении ее поведения
- *Парное программирование* – код пишется двумя программистами на одном компьютере

Практики XP

- *Коллективное владение кодом* – любой разработчик может улучшить код любого модуля системы
- *Непрерывная интеграция* – система интегрируется как можно чаще; непрерывное регрессионное тестирование гарантирует сохранение функциональности при изменении требований
- *Непрерывное тестирование* как отдельных модулей, так и системы в целом; входным критерием для написания кода является отказавший тестовый вариант
- *Реорганизация (Refactoring)* – улучшение структуры системы при сохранении ее поведения
- *Парное программирование* – код пишется двумя программистами на одном компьютере

Схема работы над проектом



User stories

- User Story - это описание того как система должна работать. Каждая User Story написана на карточке и представляет какой-то кусок функциональности системы, имеющий логический смысл с точки зрения Заказчика. Форма - один-два абзаца текста понятного пользователю (не сильно технического).
- User Story пишется Заказчиком. Они похожи на сценарии использования системы, но не ограничиваются пользовательским интерфейсом. По каждой истории пишутся функциональные тесты, подтверждающие что данная история корректно реализована - приемочные (Acceptance tests).

Планирование релиза

- Релиз - планы описывают взгляд на весь проект и используются в дальнейшем для планирования итераций.
- Планирование Релиза определяет набор правил, которые позволяют всем принимать свои решения. Эти правила определяют метод выработки удовлетворяющего всех плана работ.
- Сущность собрания по планированию релиза для команды разработчиков в том, чтобы оценить каждую User Story в идеальных неделях. **Идеальная неделя** - это сколько повашему займет время выполнение задачи, если ничто больше вас не будет отвлекать.

Итерации в схеме XP:



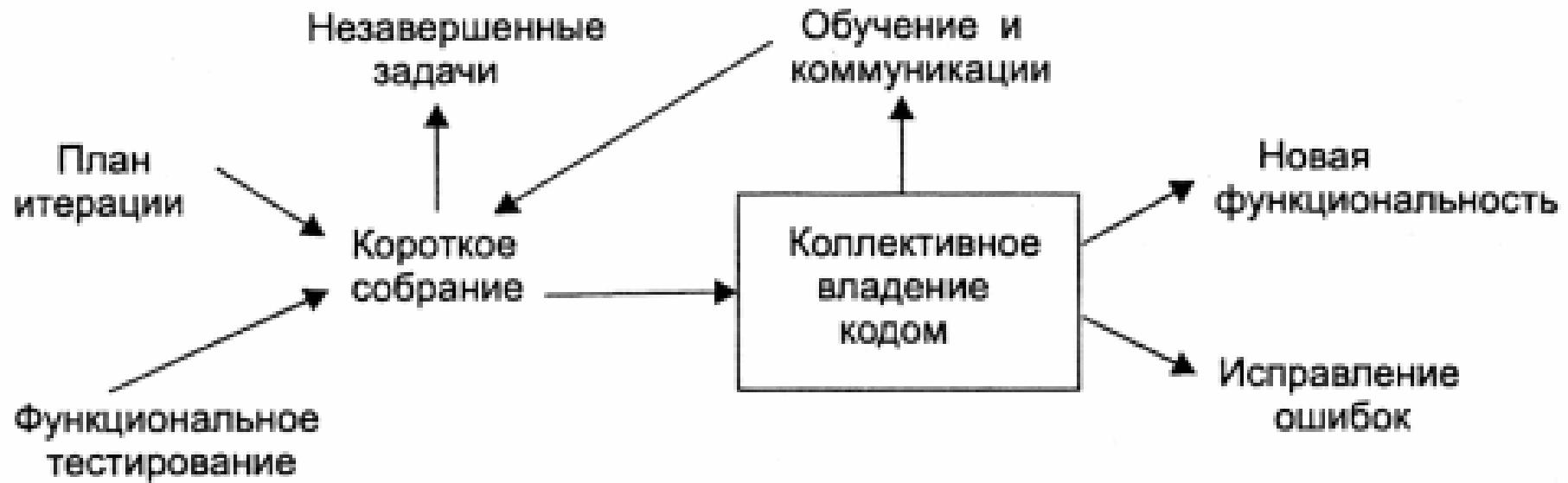
Итерации плана

Итеративная разработка увеличивает гибкость процесса.

Продолжительность итерации от 2 до 3 недель.

Цель каждой итерации - включить в версию несколько новых историй. На собрании по планированию итерации определяется, какие именно истории будут реализованы и каким образом это будет сделано командой разработчиков.

Разработка (в схеме итерации XP)



Обмен задачами

- Необходимо периодически менять задачи у разработчиков для уменьшения риска концентрации знаний и узких мест в коде. Если только один человек в команде может работать в данной области – БОЛЬШОЙ риск.
- Cross Training практика, которая позволяет избежать концентрации знаний в одном человеке. Cross Training - перемещение людей по коду в комбинации с парным программированием Вместо одного человека, который знает все о данном куске кода, каждый в команде знает много о коде в каждом модуле.

Тестирование

Любой код должен иметь **Unit Test**.

- **Unit тесты** играют ключевую роль в ХР. Они позволяют быстро менять код не боясь наделать новых ошибок. Unit тест пишется для каждого класса, тест должен проверять все аспекты работы класса - тестировать все что может не работать.
- **Важно!!!** Тест для класса должен быть написан раньше самого класса. Для проведения тестов пишется специальная система тестирования со своим интерфейсом.

Основная идея

- Экспериментальный факт: проекты, над которыми работают небольшие, кросс-функциональные команды, обычно систематически производят лучшие результаты

Роли

- Главные действующие роли:
 - ScrumMaster, тот кто занимается процессами и работает в качестве руководителя проекта,
 - Владелец Продукта, человек, который представляет интересы конечных пользователей и других заинтересованных в продукте сторон,
 - Команда, которая включает разработчиков

Этапы разработки

- Процесс разработки разбивается на отдельные этапы определенной длительности – *спринты* (обычно, 15-30 дней)
- Каждому спринту предшествует этап, который называется *product backlog* – документирование запросов на выполнение работ

Планирование спрингта

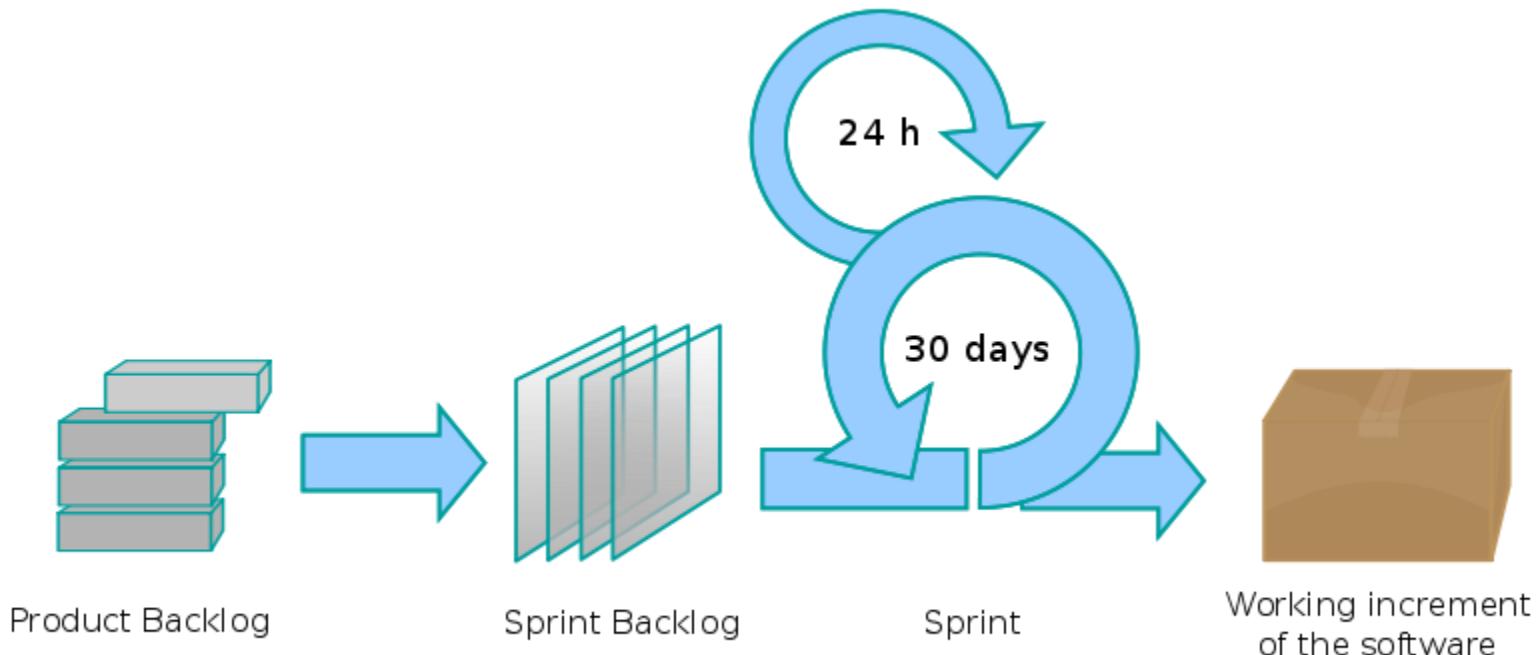


- Запросы на выполнение работ определяются на этапе совета по планированию спрингта – *sprint planning meeting*
- На протяжении этого собрания Владелец Продукта информирует о заданиях, которые должны быть выполнены
- Команда определяет, сколько из желаемого они могут выполнить, чтобы завершить необходимые части на протяжении следующего спрингта

Выполнение спрингта

- Во время спрингта команда выполняет определенный фиксированный список задач - *backlog items*, наращивая функциональность программного продукта
- На протяжении этого периода никто не имеет права менять список требований к работе, что следует понимать, как заморозку требований (*requirements*) во время спрингта

Scrum



Business modeling (бизнес-анализ)

- **Артефакты-модели:**

- модель бизнес-процессов - определение бизнес-требований к разрабатываемой системе;
- модель структуры предприятия - артефакт для разработки функциональной модели системы;
- модели документов, бизнес-сущностей, модели сценариев бизнес-функций, модели состояний бизнес-сущностей - для проектирования пользовательского интерфейса, БД системы; представляют собой описание статического и динамического состояний системы с различных точек зрения;
- модели бизнес-правил - артефакт используется для моделирования правил в ПО.
- план работ на этапе бизнес-моделирования;
- рекомендации по проведению бизнес-моделирования;
- запросы на изменение.

Business modeling (бизнес-анализ)

- **Артефакты-документы:**

- оценка организации заказчика, структура бизнеса;
- словарь терминов предметной области;
- набор бизнес-правил;
- коммерческое предложение;
- спецификации бизнес-функций;
- план работ на этапе бизнес-моделирования;
- рекомендации по проведению бизнес-моделирования;
- запросы на изменение.

Requirements (требования)

- **Артефакты-модели:**
 - модель функции системы;
 - модель сценариев функций системы;
 - модель интерфейсов пользователя;
 - модель сценариев работы пользователя системы;
 - модель выходных форм;
 - модель правил системы.
 - запросы на изменение.

Requirements (требования)

- **Артефакты-документы:**

- план управления требованиями;
- словарь терминов системы;
- спецификация на программную систему;
- спецификация на функции системы;
- правила системы;
- запросы заинтересованных лиц;
- план работ на этапе определения требований к системе;
- рекомендации по моделированию на этапе определения требований;
- запросы на изменение.

Analysis and design (анализ и проектирование)

- **Артефакты-модели:**
 - логическая модель данных;
 - физическая модель данных;
 - модель спецификаций компонентов системы;
 - сценарии взаимодействия классов, реализующих компоненты системы.
- **Артефакты-документы:**
 - архитектура программного обеспечения;
 - спецификации программных компонентов;
 - рекомендации на этапе анализа и проектирования;
 - план работ на этапе анализа и проектирования;
 - запросы на изменение.

Implementation (реализация, кодирование)

- **Артефакты-модели:**
 - компонентная модель приложения.
- **Артефакты-код:**
 - элементы генерации кода, полученные в Rational Rose;
 - собственно код приложения;
 - документация.
- **Артефакты-документы:**
 - план сборки приложения;
 - план работ на этапе реализации.

Test (тестирование)

- **Артефакты-модели:**

- модель тестовых примеров;
- функциональная модель тестовой программы;
- модель спецификации компонентов тестовой программы;
- сценарии взаимодействия классов, реализующих взаимодействие компонентов тестовой программы.

- **Артефакты-документы:**

- описание тестовых примеров;
- план тестирования;
- план работ на этапе тестирования;
- запросы на изменение.

Deployment (внедрение)

- **Артефакты-модели:**
 - модель размещения - описание размещения компонентов по узлам обработки.
- **Артефакты-документы:**
 - обучающие материалы;
 - документы по инсталляции;
 - описание версий системы;
 - план внедрения

UML (Unified Modeling Language)

- Унифицированный язык моделирования UML (Unified Modeling Language) - это графический язык визуализации спецификации и документирования артефактов преимущественно программной системы.
- Язык UML
 - представляет собой стандартное средство создания чертежной системы
 - определяет конкретные понятия
 - классы, написанные на определенных языках программирования,
 - схемы баз данных
 - программные компоненты с возможностью повторного использования.
 - позволяет разработчикам определять, визуализировать, конструировать и документировать артефакты программных систем.

SCRUM-МОДЕЛЬ

RAD-МОДЕЛЬ

RAD-модель

- Модель быстрой разработки приложений (Rapid Application Development) является примером адаптивного процесса в рамках реализации инкрементной стратегии
- Основателем RAD считается сотрудник IBM Джеймс Мартин, который в 1980-х годах сформулировал основные принципы RAD, основываясь на идеях Барри Бойема и Скотта Шульца

Цели RAD-модели

- Основными целями RAD-модели процесса разработки ПО являются:
 - высокая скорость разработки;
 - низкая стоимость;
 - высокое качество



Основные принципы RAD

1. Работа ведется группами; типичный состав группы - руководитель, аналитик, два программиста, технический писатель.
2. Разработка базируется на моделях; моделирование позволяет оценить проект и выполнить его декомпозицию на составные части, каждая из которых может разрабатываться отдельной RAD-группой.

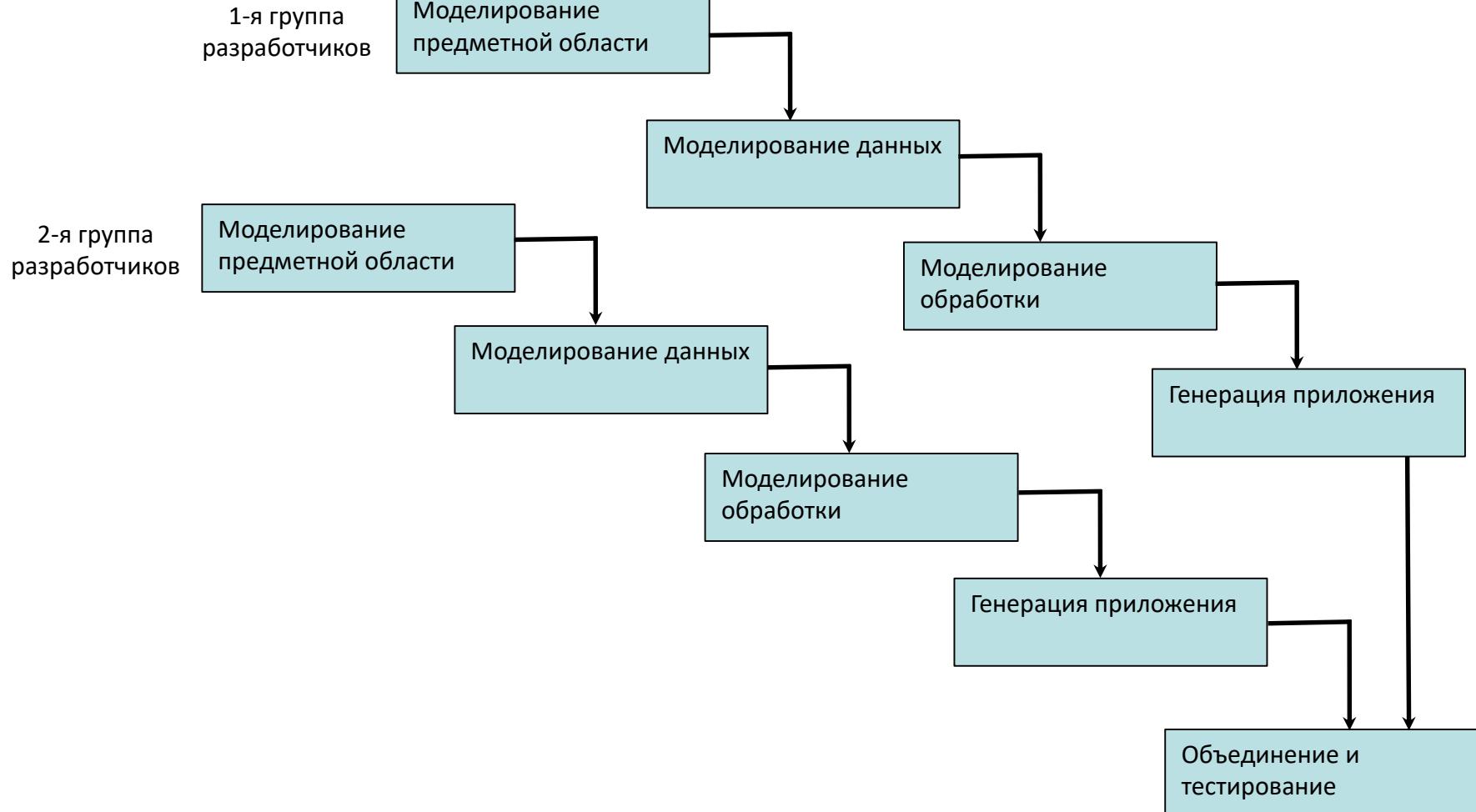
Основные принципы RAD

3. Разработка системы и предъявление ее заказчику осуществляется в виде последовательности развиваемых прототипов
4. RAD-группа всегда работает только над одним прототипом. Это обеспечивает единство целей, лучшую наблюдаемость и управляемость процессом разработки

Основные принципы RAD

5. RAD-группы должны использовать общие стандарты
6. Обязательно финальное тестирование полной системы
7. Обязательно использование инструментальных средств, автоматизирующих процесс разработки – визуальных сред проектирования и программирования

RAD-модель



Прототипы

- Любой из прототипов реализует определенную часть функциональности, требуемой от конечного продукта; каждый последующий прототип включает всю функциональность, реализованную в предыдущем прототипе, с добавлением новой
- Число прототипов определяется на основе учета разных параметров – размера проекта, анализа рисков, пожеланий заказчика и т. д.

Прототипы

- Традиционно для проектов ПО средней сложности разрабатываются три прототипа:
 - первый содержит весь пользовательский интерфейс с нулевой функциональностью; он дает возможность утвердить у заказчика экранные и отчетные формы;
 - второй прототип содержит реализованную на 70-80% функциональность системы
 - третий прототип содержит полностью реализованную функциональность

Итерации

- Основаниями для очередной итерации в процессе разработки являются:
 - Замечания заказчика. Если замечания носят характер исправлений, они учитываются в следующем прототипе, если же изменяются требования, то выполняется переоценка проекта и корректируются сроки и стоимость проекта

Итерации

- *Детализация.* Выполняется программирование нереализованной части системы в соответствии с составленным планом.
- *Анализ результатов программирования.* Исправляются ошибки, повышается эффективность программного кода и т. д.

Когда применяется RAD

- Применение технологии RAD целесообразно, когда:
 - требуется выполнение проекта в сжатые сроки (90 дней); быстрое выполнение проекта позволяет создать систему, отвечающую требованиям сегодняшнего дня
 - нечетко определены требования к ПО; в большинстве случаев заказчик весьма приблизительно представляет себе работу будущего программного продукта и не может четко сформулировать все требования к ПО

Когда применяется RAD

- проект выполняется в условиях ограниченности бюджета; разработка ведется небольшими RAD-группами в короткие сроки, что обеспечивает минимум трудозатрат и позволяет вписаться в бюджетные ограничения
- интерфейс пользователя (GUI) есть главный фактор; RAD-технология дает возможность продемонстрировать интерфейс в прототипе, причем достаточно скоро после начала проекта

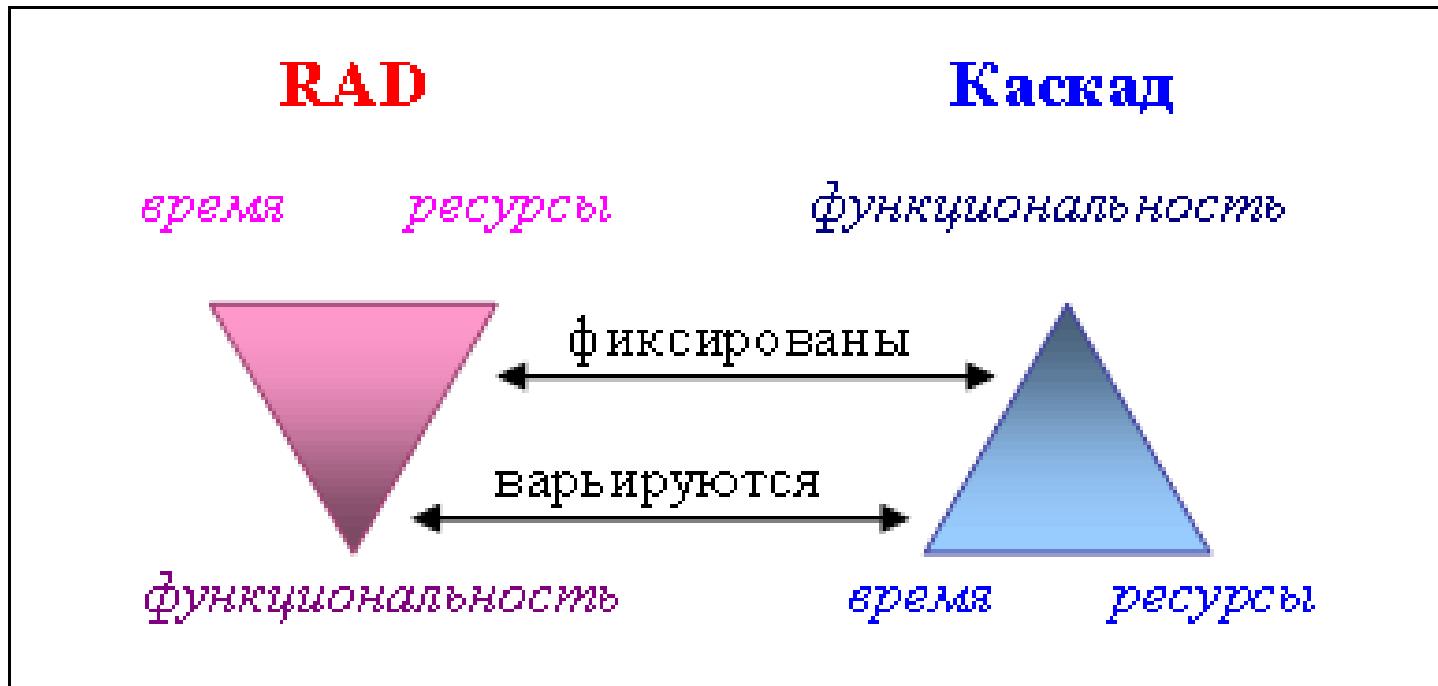
Когда применяется RAD

- проект большой, но поддается разделению на более мелкие функциональные компоненты
- ПО не обладает большой вычислительной сложностью

RAD не применяется

- В проектах, где требования к программному продукту четко определены и не должны меняться, и, следовательно, вовлечение заказчика в процесс разработки не требуется
- В проектах, сложность которых определяется необходимостью реализации сложных алгоритмов, а роль и объем пользовательского интерфейса невелик

Сравнение двух моделей



Характеристика модели

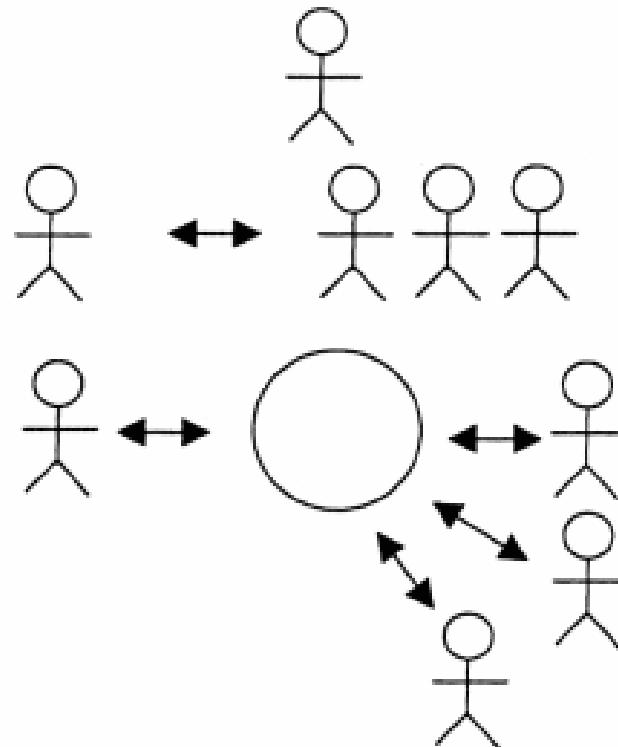
- Основным достоинством модели является уменьшение сроков разработки
- Ее главный недостаток заключается в необходимости использования большого числа квалифицированных разработчиков, что может существенно повысить стоимость разработки

ОРГАНИЗАЦИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1. Группа проекта и роли участников
2. Документация процесса разработки программного обеспечения.
- 3.

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Триада разработок в зависимости от количества участников



1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Авторская разработка

Жизненный цикл разработки поддерживается одним единственным человеком (70-80-е годы ХХ века).

Применение в области наукоемких приложений:

- необходимость многолетнего изучения предметной области,
- практически полное отсутствие начального финансирования проекта,
- малая рентабельность, определяемая узким кругом пользователей.

Объем программного продукта, выполненного методом авторской разработки, в 5-20 раз меньше по сравнению с индустриальными аналогами.

Авторская разработка предполагает достижение профессионального успеха и известности .

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Авторская разработка

Иногда авторская разработка может выигрывать по производительности в тридцать и более раз у коллективной разработки, что достигается за счет:

- исключения межличностных коммуникаций, связанных с необходимостью порождения и изучения большого количества технологической документации;
- исключения работ по разбиению проекта на составляющие, по распределению их между исполнителями, по координации деятельности исполнителей и контролю за их работой.

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

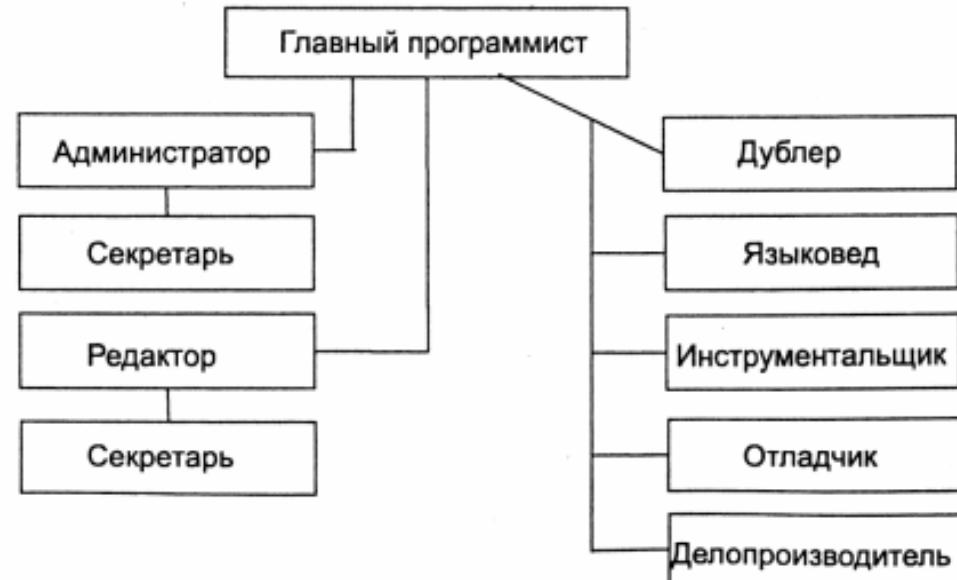
Коллективная разработка

Собрать стадо из баранов легко, трудно собрать стадо из кошек.

С. П. Капица

Одним из основных вопросов коллективной разработки является разделение труда - от равноправных соисполнителей до организации в виде жесткой структуры.

Харлан Миллз [Брукс 1999] предложил организовывать команды (бригады) главного программиста (chief programmer teams), подобные хирургическим бригадам.

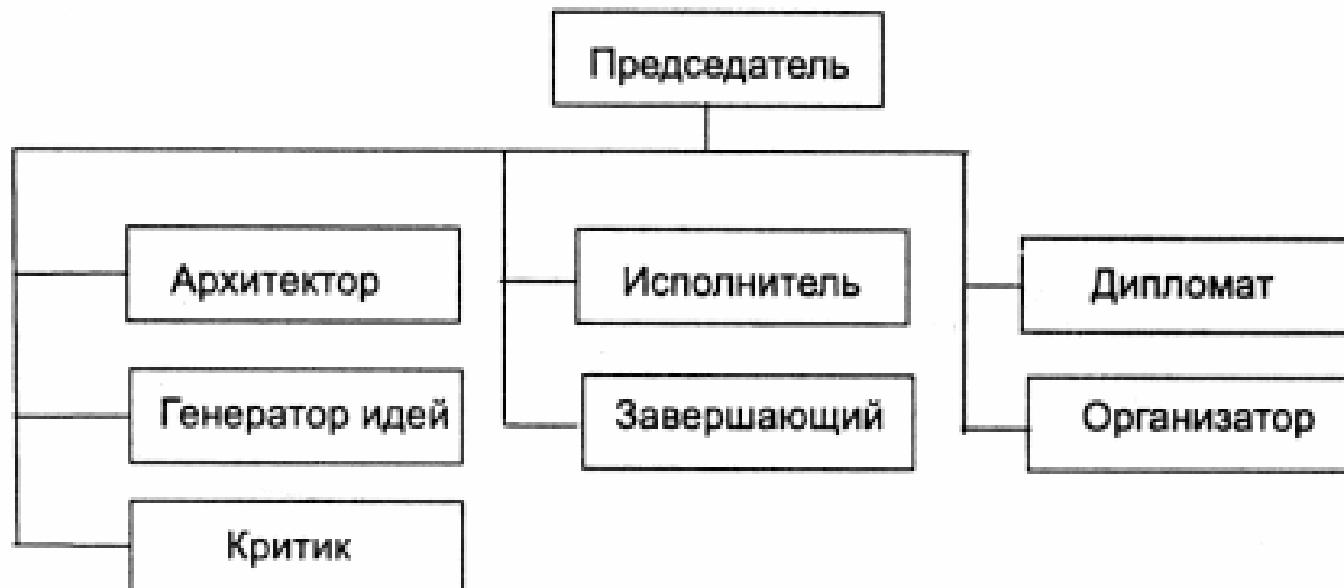


1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Коллективная разработка

Психологические командные роли

Роб Томсет (Rob Thomsett) [Thomsett 1990] предложил восемь ключевых ролей в проекте



1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Коллективная разработка

Психологические командные роли

Роб Томсет (Rob Thomsett) [Thomsett 1990] предложил восемь ключевых ролей в проекте:

Председатель. Выбирает путь, по которому команда движется вперед к общим целям. Умеет обнаружить сильные и слабые стороны команды и обеспечить наибольшее применение потенциала каждого ее участника.

Архитектор. Придает законченную форму действиям команды. Имеет четкое представление о проблемах и их возможных решениях.

Генератор идей. Предлагает радикально новые идеи и стратегии, новые подходы к решению проблем, с которыми сталкивается группа. Особое внимание уделяет главным проблемам.

Критик. Он же скептик, оценивающий проблемы с прагматической точки зрения. Ищет недостатки, изъяны и недоделки. Компенсирует оптимизм генератора идей.

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Коллективная разработка

Исполнитель. Работник, собственно занимающийся написанием кода. Как правило, он не обладает широтой кругозора.

Завершающий. Поддерживает в команде настойчивость в достижении цели. Играет доминирующую роль на завершающих стадиях разработки.

Дипломат. Поддерживает силу духа в участниках проекта. Оказывает им помощь в трудных положениях. Пытается улучшить взаимоотношения в команде.

Организатор. Обнаруживает и сообщает о новых идеях, разработках и ресурсах. Имеет много друзей и связей в своей организации, с помощью которых можно выпросить или одолжить необходимые ресурсы.

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Общинная разработка

Совершенство в проекте достигается не тогда, когда нечего добавить, а тогда, когда нечего убрать.

Антуан де Сент-Экзюпери

Идеология общинной ("базарной") модели разработки сформулирована в программной статье Эрика Раймонда "Собор и Базар".

Характеристики:

- Децентролизованность разработки. Не существует ограничения сверху на количество людей, принимающих участие в проекте.
- Разработка ведется на базе открытых исходных текстов.
- Большое количество внешних тестеров (бета-тестеров), позволяющих быстро обнаруживать ошибки и проблемы в программе.

2. ДОКУМЕНТАЦИЯ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Роль документации в проекте:

- средство передачи информации между разработчиками ПС,
- средство управления разработкой ПС,
- средство передачи пользователям информации, необходимой для применения и сопровождения ПС.
- .
- Документацию процесса разработки можно разбить на две группы:
- 1.Документы управления разработкой ПС
- 2. Документы, входящие в состав ПС (product documentation)
- .

2. ДОКУМЕНТАЦИЯ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Документы управления разработкой ПС:

- Планы, Оценки, Расписания
- Отчёты
- Стандарты
- Рабочие документы
- Заметки и переписка

2. ДОКУМЕНТАЦИЯ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Документы, входящие в состав ПС

1.Пользовательская документация ПС.

- Общее функциональное описание ПС
- Руководство по инсталляции ПС
- Инструкция по применению ПС
- Справочник по применению ПС
- Руководство по управлению ПС

2.Документация по сопровождению ПС

- Документация, определяющая строение программ и структур данных ПС и технологию их разработки
- Документация, помогающая вносить изменения в ПС

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

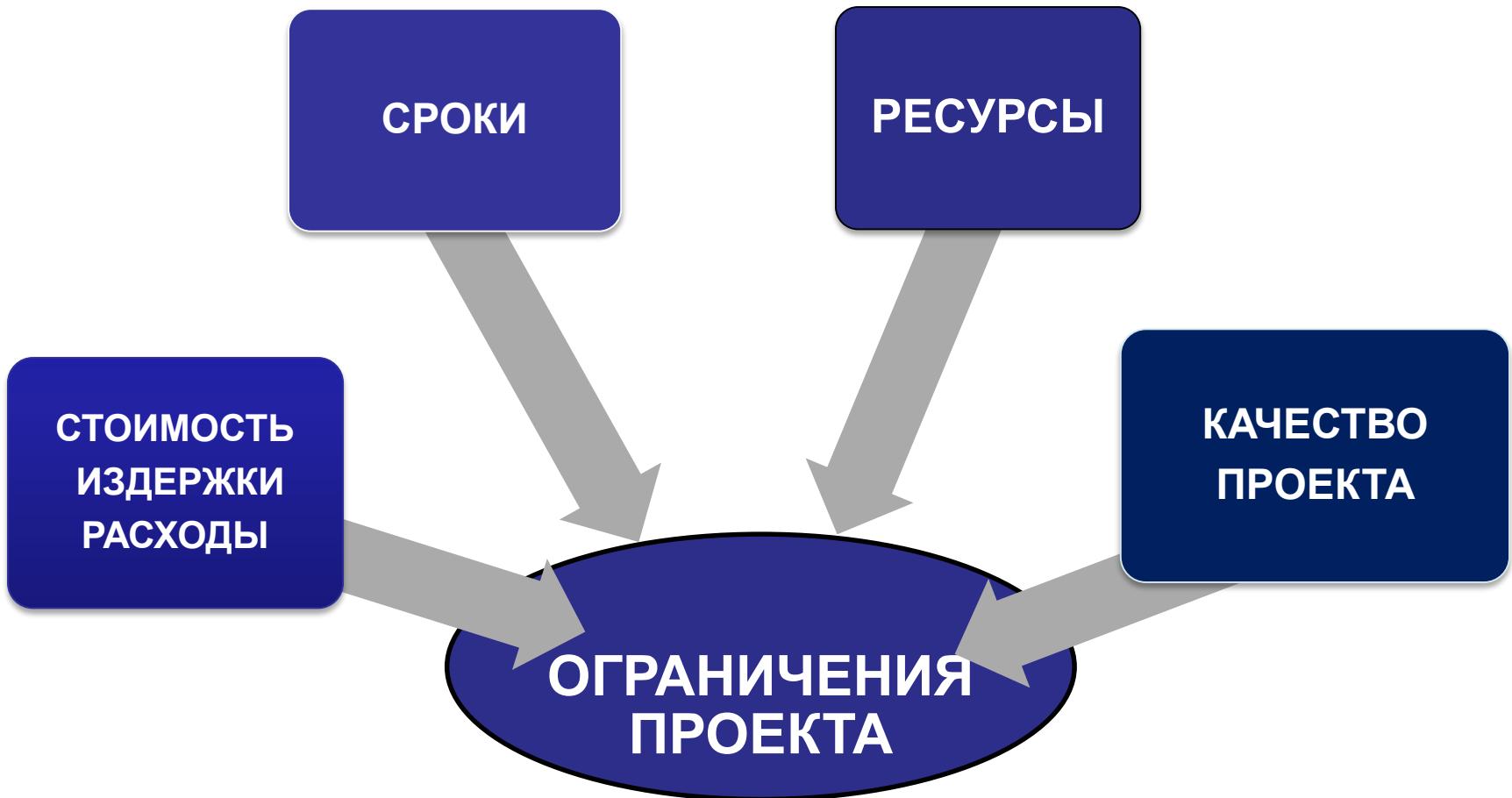
Составляющие управления проектом

- **Проект** – целенаправленное ограниченное во времени мероприятие, направленное на создание уникального продукта или услуги.
- **Управление проектом** заключается в управлении производством программного продукта в рамках отведенных средств и времени.

PMBOK. Руководство к Своду знаний по управлению проектами, 3-е изд., PMI, 2004.

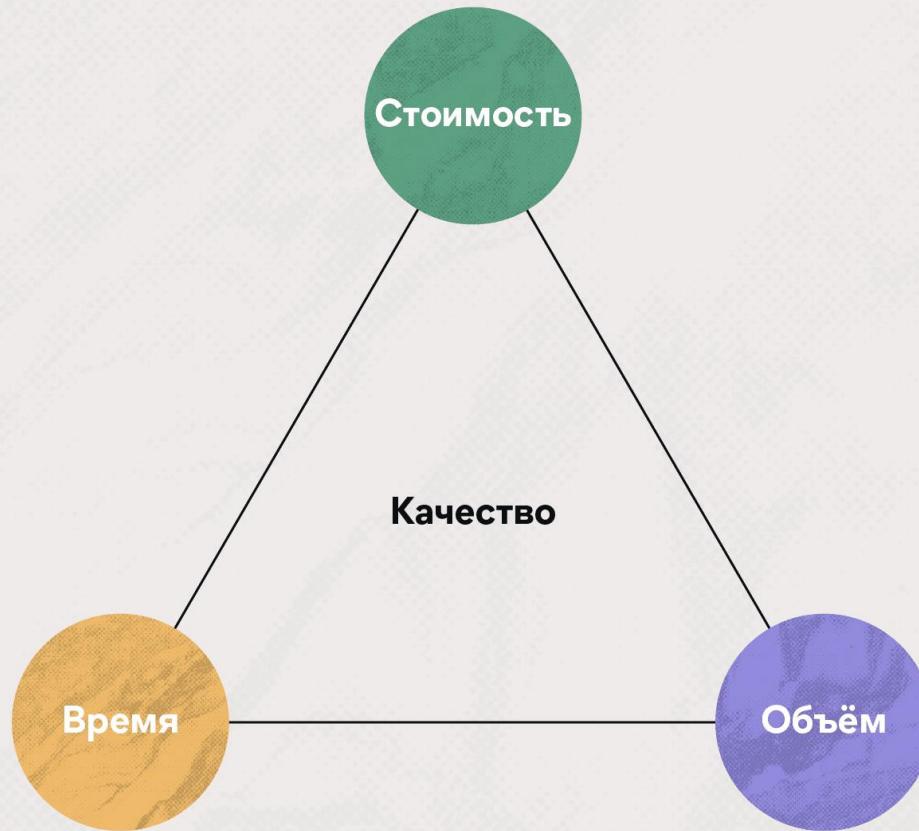
PMI, Project Management Institute, PMBOK — американский национальный стандарт ANSI/PMI 99-001-2004.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ



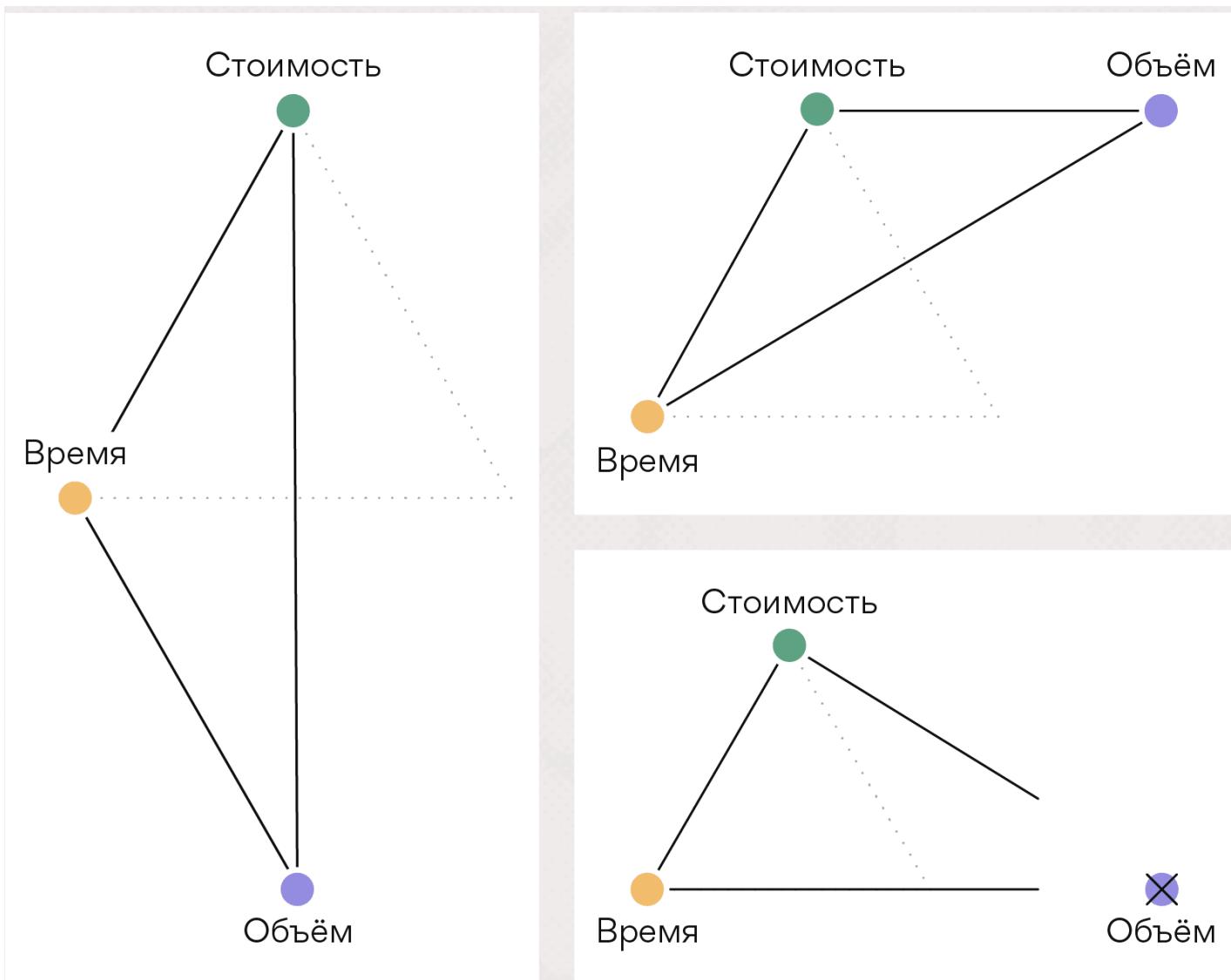
3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Железный треугольник управления проектами



3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Проектный треугольник



3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Составляющие управления проектом

Управление проектом охватывает:

- инфраструктуру (организационные моменты);
- управляющий процесс (права и ответственные участники);
- процесс разработки (методы, инструменты, языки, документация и поддержка);
- расписание (моменты времени, к которым должны быть представлены выполненные фрагменты работы).

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

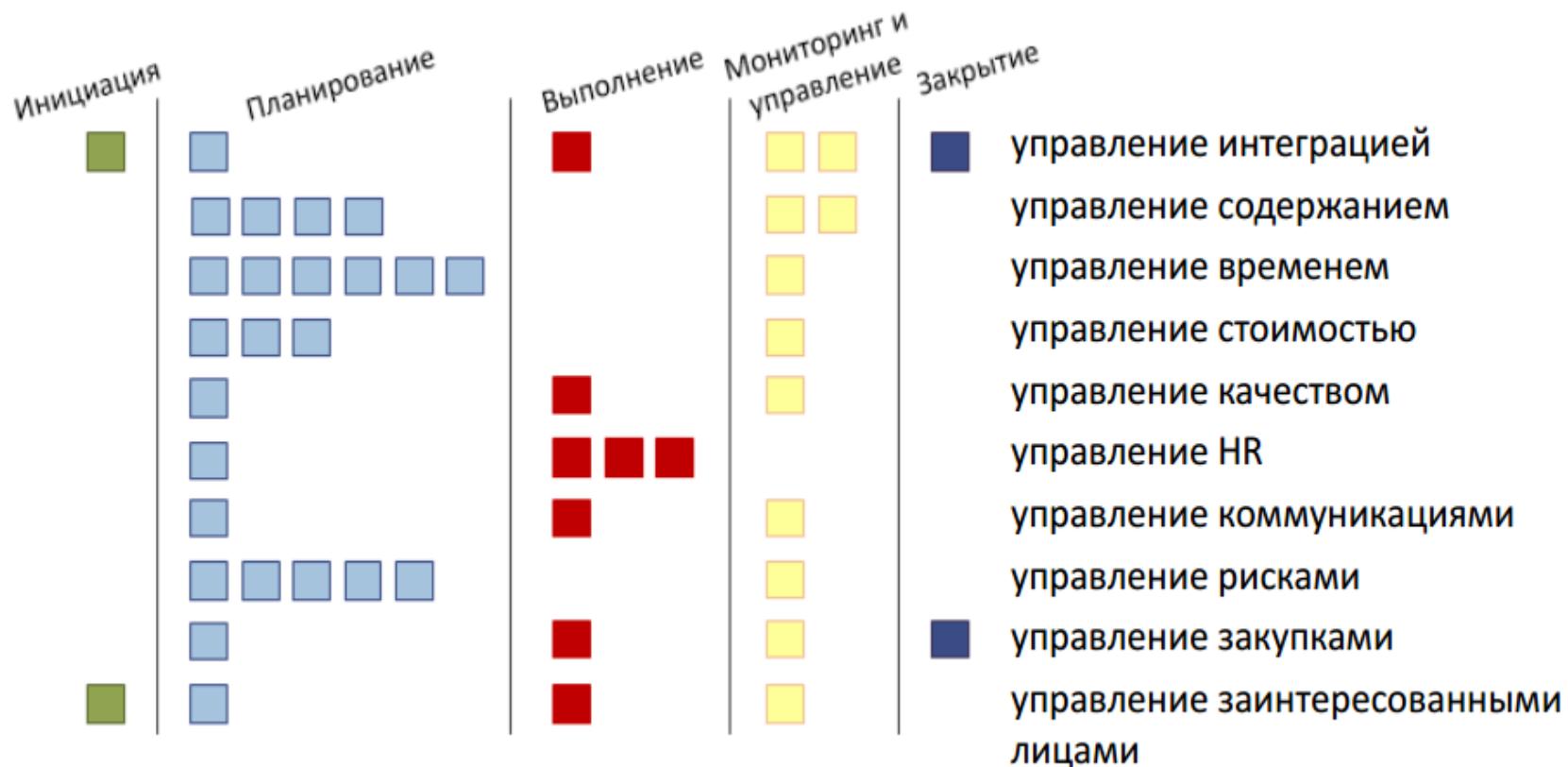
Составляющие управления проектом

Типичная схема процесса управления проектом

1. Понять содержание проекта, область применения и временные рамки.
2. Определиться с процессом разработки.
3. Выделить организационную структуру.
4. Определить управляющий процесс (ответственность участников).
5. Разработать расписание проекта.
6. Разработать план подбора кадров.
7. Начать управление рисками.
8. Определить, какие документы необходимо выработать.
9. Начать процесс разработки.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Структура стандарта



3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Соответствие между процессами и областями знаний

Область знаний	Группа процессов инициации	Группа процессов планирования	Группа процессов исполнения	Группа процессов мониторинга и контроля	Группа процессов закрытия
1. Управление интеграцией проекта	1.1. Разработка устава проекта	1.2. Разработка плана управления проектом	1.3. Руководство и управление работами проекта	1.4. Мониторинг и контроль работ проекта 1.5. Интегрированный контроль изменений	1.6. Закрытие проекта или фазы
2. Управление содержанием проекта		2.1. Планирование управления содержанием 2.2. Сбор требований 2.3. Определение содержания 2.4. Создание ИСР		2.5. Подтверждение содержания 2.6. Контроль содержания	
3. Управление сроками проекта		3.1. Планирование управления расписанием 3.2. Определение операций 3.3. Определение последовательности операций 3.4. Оценка ресурсов операций 3.5. Оценка длительности операций 3.6. Разработка расписания		3.7. Контроль расписания	
4. Управление стоимостью проекта		4.1. Планирование стоимостью проекта 4.2. Оценка стоимости 4.3. Определение бюджета		4.4. Контроль стоимости	
5. Управление закупками проекта		5.1. Планирование управления закупками	5.2. Проведение закупок	5.3. Контроль закупок	5.4. Закрытие закупок

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Области знаний

Область знаний	Группа процессов инициации	Группа процессов планирования	Группа процессов исполнения	Группа процессов мониторинга и контроля	Группа процессов закрытия
6. Управление коммуникациями проекта		6.1. Планирование управления коммуникациями проекта	6.2. Управление коммуникациями	6.3. Контроль коммуникаций	
7. Управление заинтересованными сторонами проекта	7.1. Определение заинтересованных сторон	7.2. Планирование управления заинтересованными сторонами	7.3. Управление вовлечением заинтересованных сторон	7.4. Контроль вовлечение заинтересованных сторон	
8. Управление человеческими ресурсами проекта		8.1. Планирование управления человеческими ресурсами	8.2. Набор команды проекта 8.3. Развитие команды проекта 8.4. Управление команды проекта		
9. Управление качеством проекта		9.1. Планирование управления качеством	9.2. Обеспечение качества	9.3. Контроль качества	
10. Управление рисками проекта		10.1. Планирование управления рисками 10.2. Идентификация рисков 10.3. Качественный анализ рисков 10.4. Количественный анализ рисков 10.5. Планирование реагирования на риски		10.6. Контроль рисков	

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Процессы управления проектами

- процессы инициации — принятие решения о начале выполнения проекта
- процессы планирования — определение целей и критериев успеха проекта, разработка плана
- процессы реализации — координация людей и других ресурсов для выполнения плана
- процессы контроля — определение соответствия плана и исполнения проекта поставленным целям и критериям успеха, принятие решений о необходимости применения корректирующих воздействий
- процессы завершения — формализация выполнения проекта и подведение его к упорядоченному финалу

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Схема жизненного цикла IT-проекта

Жизненный цикл – это последовательность фаз проекта, через которые он должен пройти для гарантированного достижения целей проекта



3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Разработка устава проекта

1. Краткое описание проекта

1.1 Название проекта

<Название проекта>

1.2 Суть проекта

<Что представляет собой проект? Это разработка или внедрение? Разработка чего и для чего – одним/двумя предложениями>

1.3 Бизнес-окружение проекта

<Почему предпринят проект; каковы ожидания / предположения высшего менеджмента? Как цели проекта связаны со стратегическими целями клиента (продукт выведет его на новые рынки, позволит сэкономить и т.п.)?

1.4. Цели проекта

<Цели проекта SMART (перечисляемым списком)>

1.5. Риски проекта

<Если известно - предполагаемые результаты предполагаемой качественной оценки совокупных рисков проекта (негативных и позитивных) – высокий / средний / низкий. Краткое обоснование оценки (1-2 предложения)>

<Если известны – главные высокоуровневые риски (перечислить)>

Устав Проекта

2. Описание продукта и поставок

2.1 Продуктом проекта является (перечень поставок)

<Перечислить результаты поставки списком - дистрибутив ПО, пользовательская документация; и т.п.>

2.2 Главными требованиями к продукту являются (продукт позволяет):

<Указать высокоуровневые требования (функциональные и нефункциональные)>

2.3 Требованиями к продукту НЕ являются (продукт не включает):

<Указать значимые высокоуровневые требования, не включаемые в настоящий проект>

2.4 Правила приемки поставок:

<Указать общие правила приемки (будет ли создаваться комиссия, на основании каких документов планируется сдача)>

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Разработка устава проекта

3. Ограничения проекта

3.1 Вехи и дата завершения проекта:

Начало проекта	<Дата>
• <Указать название вехи 1>	<Дата>
• <Указать название вехи 1>	<Дата>
• <...>	<...>
Завершение проекта	<Дата>

3.2 Общий бюджет проекта:

<Бюджет включает все расходы по проекту + все расходы по управлению рисками, в том числе и управленческие резервы>

3.3 Ограничения по выполнению и организации работ

<Например, для успеха проекта критично важно, чтобы сотрудники исполнителя не общались с определенным департаментом заказчика; или указом менеджмента должна быть выбрана конкретная аппаратная платформа>

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Разработка устава проекта

4. Руководитель проекта и его полномочия

4.1 Назначенный руководитель проекта

<ФИО руководителя проекта - представителя исполнителя, ответственного за реализацию проекта в срок, в пределах бюджета и с заданным качеством>

4.2 Полномочия руководителя проекта

<Как руководитель проектов будет формировать команду – может ли он брать любых людей, должен ли он это обосновать финансово? Будет ли у руководителя проекта помощник?>

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

5. Заинтересованные лица и ресурсы

5.1 Заказчик проекта

<ФИО, должность, и название организации>

5.2 Ключевые пользователи результатов проекта:

<Перечень лиц или организаций>

5.3 Спонсор проекта

<ФИО, должность, и название организации>

5.4 Куратор проекта

<Если определен куратор – указать. Куратор это промежуточное звено между ПМ и спонсором, уполномоченный принимать решения о выделении ресурсов и изменениях в проекте. Куратор может быть сотрудником проектного офиса (PMO), если таковой развернут в организации>

5.5 Команда проекта

<Сколько людей выделено на проект? Из каких департаментов? Какие люди / под какие работы – будут наняты извне (в штат / по субконтракту), если это известно? Какие вводные по привлекаемым субподрядчикам известны сейчас? Какие вводные по требуемой квалификации известны? Ограничен ли бюджет на привлечение сотрудников?>

5.6 Инфраструктура

<Какие требования к специальному оборудованию, используемому на проекте известны? Потребуется ли лицензионное программное обеспечение для производства продукта? Ограничен ли бюджет на инфраструктурные ресурсы?>

5.7 Соисполнители проекта

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Разработка устава проекта

6. Согласовательные подписи			
УТВЕРЖДАЮ:			
Имя	Должность	Подпись	Дата (MM/DD/YYYY)
<Устав обязательно подписывается спонсором>			

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Пример

1.1 Название проекта

1.2 Цель проекта - повышение эффективности основной производственной деятельности отдела «123».

1. 3. Дополнительными целями проекта являются:

1. 3.1.Установление долгосрочных отношений с важным заказчиком ОАО «XYZ».

1. 3. 2.Выход на новый перспективный рынок современных B2C систем.

1.4. Повышение эффективности основной производственной деятельности отдела «123» достигается :

- снижением затрат на обработку заявок
- снижением сроков обработки заявок
- повышением оперативности доступа к информации о наличии продукции

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Пример

- повышением оперативности доступа к информации о прохождении заявок
 - повышением надежности и полноты хранения информации о поступивших заявках и результатах их обработки.
- результатах их обработки.

2. Продуктами проекта являются:

- 2.1. Прикладное ПО и документация пользователей.
- 2.2. Базовое ПО.
- 2.3. Оборудование ЛВС, рабочие станции, сервера и операционно-системное ПО.
- 2.4. Проведение пуско-наладочных работ и ввод в опытную эксплуатацию.
- 2.5. Обучение пользователей и администраторов системы.
- 2.6. Сопровождение системы на этапе опытной эксплуатации.
- 2.7. Передача системы в промышленную эксплуатацию.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Пример

3. Система должна автоматизировать следующие функции:
 - 3.1. Авторизация и аутентификация пользователей.
 - 3.2. Просмотр каталога продуктов.
 - 3.3. Поиск продуктов по каталогу.
 - 3.4. Заказ выбранных продуктов.
 - 3.5. Просмотр информации о статусе заказа.
 - 3.6. Информирование клиента об изменении статуса заказа.
 - 3.7. Просмотр и обработка заказов исполнителями из службы продаж.
 - 3.8. Просмотр статистики поступления и обработки заказов за период.
 - 3.9. Подготовка и сопровождение каталога продукции.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Пример

4. В рамки проекта не входят:

4.1. Защита системы от преднамеренного взлома.

4.2.Разработка B2B API и интеграция с другими системами.

5. Риски проекта

Задачи системы поняты недостаточно полно. Понимание масштаба и рамок проекта недостаточно. Системы создаются на новой технологической платформе, сомнения в рыночной стабильности платформы. Суммарный уровень рисков следует оценить выше среднего..

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Пример

6.1. Сроки проекта

6.1.1. **03.03** старт

6.1.2 **28.11** завершение

6.1. 3 Контрольные точки:

15.04 ТЗ утверждено

30.04 1-я итерация завершена. Подсистема заказа документации передана в тестовую эксплуатацию (на серверах разработчика).

5.05 Монтаж оборудования у заказчика завершен....

.....

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Пример

6.2. Расходная часть бюджета проекта

6.2.1. Разработка и сопровождение прикладного ПО:

6.2.1.1. $9000 \text{ чел.} * \text{час.} * \$40 = \$360\,000$

6.2.2. Поставка оборудования и операционно-системного ПО:

6.2.2.1. Сервера $* \$10\,000 = \$30\,000$

6.2.2.2. Поставка базового ПО:

Oracle RDBMS \$20 000

.....

Итого:

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Ключевые участники и заинтересованные стороны

- Спонсор проекта - лицо или группа лиц, предоставляющая финансовые ресурсы для проекта в любом виде.
- Заказчик проекта - лицо или организация, которые будут использовать продукт, услугу или результат проекта.
- Пользователи результатов проекта.
- Куратор проекта - представитель исполнителя, уполномоченный принимать решение о выделении ресурсов и изменениях в проекте
- Руководитель проекта - представитель исполнителя, ответственный за реализацию проекта в срок, в пределах бюджета и с заданным качеством.
- Соисполнители проекта. Субподрядчики и поставщики

Процессы планирования

Описание продукта

Устав проекта

Создание WBS

Содержание проекта

Иерархическая структура работ (WBS)

Создание расписания

Сетевая диаграмма

Пул ресурсов

Расписание проекта

Создание Плана проекта

Бюджет проекта

Планы управления сроками,
стоимостью ...

WBS (*Work Breakdown Structure*) - Иерархическая структура
работ

Определение содержания

Декомпозиция – последовательное деление основных результатов проекта на меньшие компоненты до уровня детализации, достаточного для обеспечения управления проектом (планирования, исполнения, контроля и завершения)

Корректность декомпозиции

- Каждый элемент **полностью определен**
- Нижнеуровневые операции и **необходимы, и достаточны для соответствующего элемента**
- Каждый элемент нижнего уровня **может быть оценен по** стоимости, срокам и приписан к организационной единице (подразделение, исполнитель), которая способна его выполнить

Определение содержания Декомпозиция - метод



- Каждая цель имеет свой код в иерархии дерева целей.
- Каждая работа имеет свой код в иерархии процесса.
- Лучше планировать цели так, чтобы подзадач было не более 7 (5+2).

WBS ур. 1

1 Велосипед 100

WBS ур. 2

- 1.1 Набор для рамы 15
- 1.2 Ведущие звёзды 5
- 1.3 Колёса 30
- 1.4 Тормозная система 5
- 1.5 Переключ. скоростей 5
- 1.6 Сборка 35
- 1.7 Управл-е проектом 5

WBS ур. 3

1.1 Набор для рамы

- 1.1.1 Рама 7
- 1.1.2 Руль 2
- 1.1.3 Вилка 3
- 1.1.4 Сиденье 3

1.2 Ведущие звёзды 5

1.3 Колёса

- 1.3.1 Пер. колесо 13
- 1.3.2 Зад. колесо 17

1.4 Тормозная система 5

1.5 Переключ. скоростей 5

1.6 Сборка

- 1.6.1 Концепт 3
- 1.6.2 Дизайн 5
- 1.6.3 Монтаж 10
- 1.6.4 Тестирование 17

1.7 Управл-е проектом 5

Планирование работ

- *Иерархическая структура работ (work breakdown structure)* – иерархическое разбиение всей работы, которую необходимо выполнить для достижения целей проекта, на более мелкие операции, до такого уровня, на котором способы выполнения этих действий вполне ясны и соответствующие работы могут быть оценены и спланированы.
- Операция – работа, задача

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Планирование работ

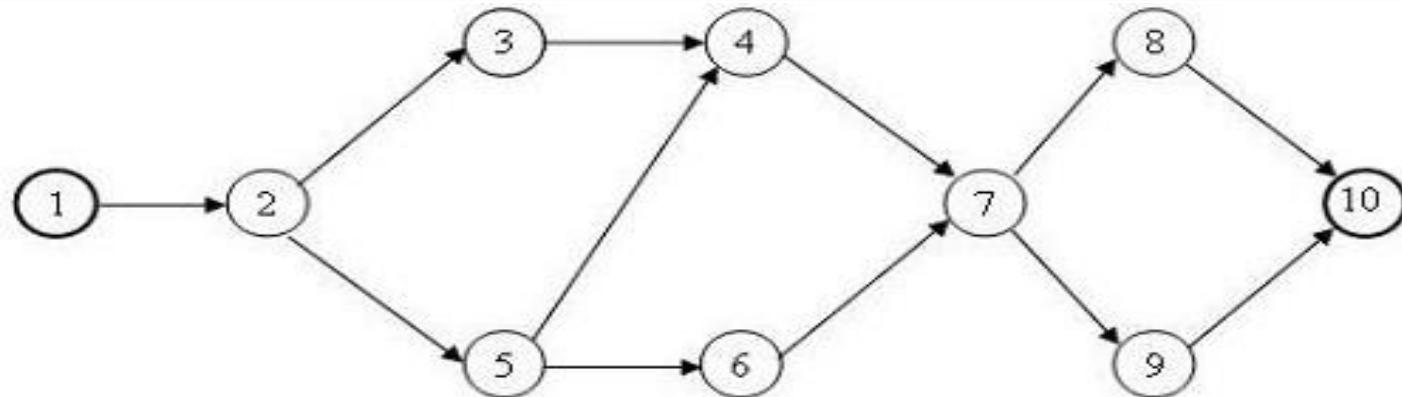
- ***Разработка графика проекта***

- Уточнить имеющуюся структуру работ проекта
- Установить зависимости между отдельными работами
- Оценить время выполнения и трудоемкость каждой из работ
- Построить *сетевой график* проекта

Сетевой график – это ориентированный граф, в котором вершинами обозначены работы проекта, а дугами – временные взаимосвязи работ.

- Выделить *критический путь* — последовательностей работ от начала до конца проекта, сумма длительностей которых максимальна среди таких последовательностей.

Номер работы	Название работы	Длительность
1	Начало реализации проекта	0
2	Постановка задачи	10
3	Разработка интерфейса	5
4	Разработка модулей обработки данных	7
5	Разработка структуры базы данных	6
6	Заполнение базы данных	8
7	Отладка программного комплекса	5
8	Тестирование и исправление ошибок	10
9	Составление программной документации	5
10	Завершение проекта	0



3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Планирование работ

- ***Разработка графика проекта***
 - Построить календарный график проекта (диаграмму Ганта), который отражает:
 - структуру работ, полученную на основе сетевого графика;
 - состав используемых ресурсов и их распределение между работами;
 - календарные даты, к которым привязываются моменты начала и завершения работ.

Последовательность операций

- **Последовательность операций** - идентификация и документирование логических взаимосвязей между операциями

Входы

- Перечень операций
- Зависимости
- **Контрольные события (milestones)** – операции с в результате исполнения которых достигаются промежуточные цели

Методы

- Диаграммные методы

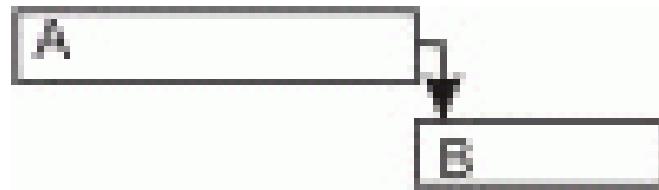
Результаты

- Сетевая диаграмма проекта

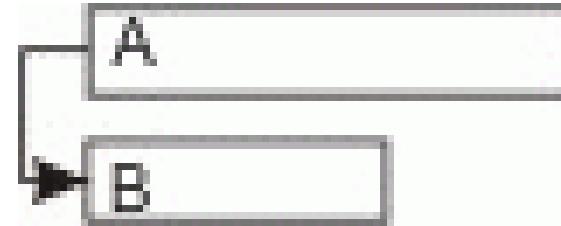
3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Типы зависимостей

- Finish-to-start (Окончание-начало), или сокращенно FS (ОН), задача В не может начаться, пока не завершена задача А.



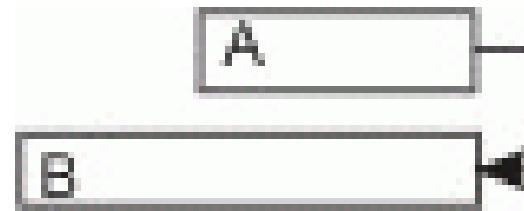
- Start-to-start (Начало-начало), или сокращенно SS (НН), задача В не может начаться до тех пор, пока не началась задача А.



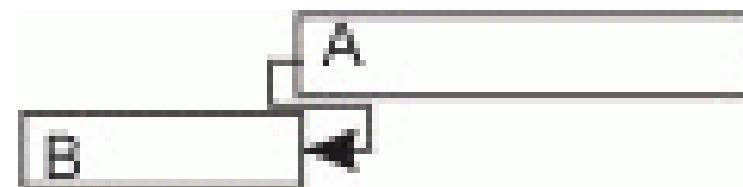
3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Типы зависимостей

- Finish-to-Finish (Окончание-окончание), или сокращенно FF (00), задача В не может закончиться до тех пор, пока не закончилась задача А.



- Start-to-Finish (Начало-окончание), или сокращенно SF (HO), задача В не может закончиться до тех пор, пока не началась задача А.



Ресурсы в проекте

Ресурс – это трудовая, материальная, финансовая, техническая или иная единица, которая используется для выполнения задач проекта.

Виды ресурсов:

- *Трудовые* – возобновляемые ресурсы (исполнители, машины и оборудование)
- *Материальные* – не возобновляемые ресурсы (материалы и энергоносители)
- *Затратные* – различные виды денежных расходов сопряженных с работами проекта, которые напрямую не зависят от объема и длительности работ.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Характеристики ресурсов

- Доступность
- Стоимость:
 - Стандартная ставка
 - Стоимость использования
- *Универсальный ресурс* (прототип ресурса) используется для определения требований к персоналу для проекта, например к тестировщикам и разработчикам.
- *Выделенный ресурс* – ресурс, формально выделенный для любого назначения задач, имеющегося в проекте (по умолчанию). Выбор данного типа резервирования влияет на доступность и загрузку ресурса.
- *Предложенный ресурс* - ресурс, ожидающий выделения ресурсов для еще не утвержденного назначения задачи. Такое назначение ресурса не уменьшает его доступности для работы по другим проектам.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Характеристики операций

- ▶ *Длительность задачи (Duration)* — это период рабочего времени, который необходим для выполнения задачи.
- ▶ *Трудозатраты (Work)* — время, затраченное сотрудниками на получение результата.

Назначение

- **Назначение** – это сопоставление задаче перечня трудовых, материальных или затратных ресурсов, которые будут задействованы при ее выполнении.
- **Объем назначения** – доля (процент) рабочего времени по индивидуальному календарю ресурса.

Трудозатраты задачи

$$T = L \cdot \sum V \cdot H$$

где L – длительность задачи, V – объем назначений ресурса, H – ежедневная длительность работы ресурса в часах, сумма берется по всем назначенным задаче трудовым ресурсам.

Типы задач

- ***Fixed Duration*** (Фиксированная длительность, ФД)
- ***Fixed Units*** (Фиксированный объем ресурсов, ФОР)
- ***Fixed Work*** (фиксированные трудозатраты, ФТ)

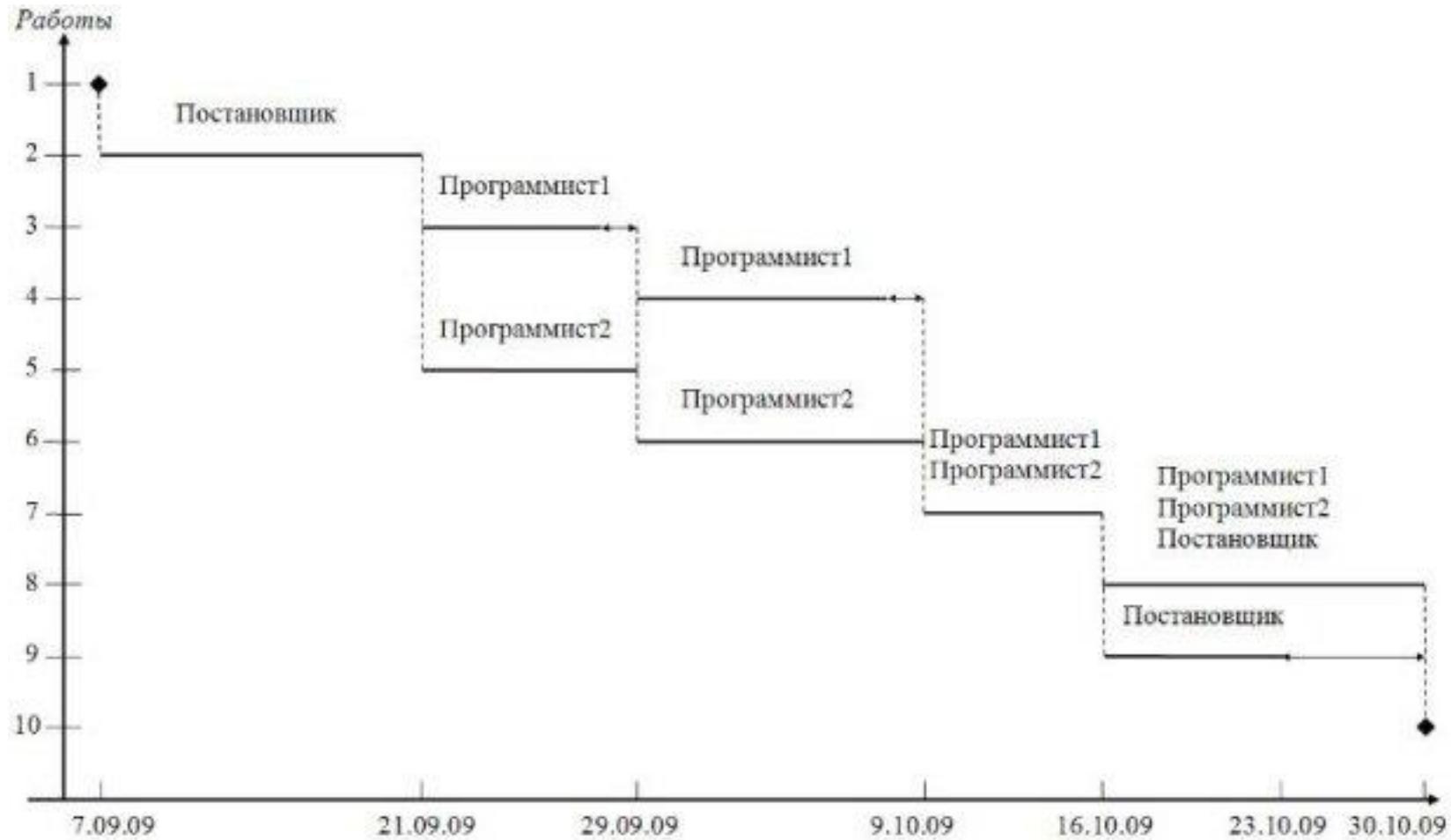
Взаимосвязь свойств для задач разных типов

	Длительности	Трудозатрат	Объема назначения ресурса	Состава ресурсов
Изменяется				
ФОР	Трудозатраты	Длительность	Длительность	Трудозатраты
ФД	Трудозатраты	Объем назначения	Трудозатраты	Трудозатраты
ФТ	Объем назначения	Длительность	Длительность	Длительность

Фиксированный объем работ

	Длительности	Трудозатрат	Объема назначения ресурса	Состава ресурсов
Изменяется				
ФОР	Трудозатраты	Длительность	Длительность	Длительность
ФД	Трудозатраты	Объем назначения	Трудозатраты	Трудозатраты

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ



Календарный график проекта

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Планирование работ

№	Название задачи	Длительность	Начало	Окончание	Пред.	28 Июл '08	04 Авг '08	11 Авг '08
						П	В	С
1	Начало реализации проекта	0 дней	Ср 30.07.08	Ср 30.07.08				
2	■ Программирование	3 дней?	Ср 30.07.08	Пт 01.08.08				
3	Постановка задачи	1 день?	Ср 30.07.08	Ср 30.07.08	1			
4	Разработка интерфейса	1 день?	Чт 31.07.08	Чт 31.07.08	3			
5	Разработка модулей обработки данных	1 день?	Пт 01.08.08	Пт 01.08.08	4;6			
6	Разработка структуры базы данных	1 день?	Чт 31.07.08	Чт 31.07.08	3			
7	Заполнение базы данных	1 день?	Пт 01.08.08	Пт 01.08.08	6			
8	Программирование завершено	0 дней	Пт 01.08.08	Пт 01.08.08	5;7			
9	■ Отладка	5 дней?	Пн 04.08.08	Пт 08.08.08				
10	Отладка программного комплекса	1 день?	Пн 04.08.08	Пн 04.08.08	8			
11	Тестирование и исправление ошибок	1 день?	Пт 08.08.08	Пт 08.08.08	100%			
12	Составление программной документации	1 день?	Пт 08.08.08	Пт 08.08.08	10			
13	Отладка завершена	0 дней	Пт 08.08.08	Пт 08.08.08	11;1			
14	Конец проекта	0 дней	Пт 08.08.08	Пт 08.08.08	13			

The Gantt chart illustrates the project timeline. Phase 1 (July 30 - August 1) includes tasks 1 through 8. Phase 2 (August 4 - 8) includes tasks 9 through 13. Task 14 is at the end. Milestones are marked with diamonds: 30.07, 01.08, 04.08, and 08.08.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

- **Риски** проекта – неопределенности, негативно влияющие на проект, вплоть до полного провала.

Риски проекта, влияющие на его ход:

- **Технологические риски** — недостаточная производительность и гибкость используемых технологий и инструментов.
- **Кадровые риски** — вероятность не набрать команду или набрать неподходящую, возможность отсутствия у ее членов необходимых навыков, возможность их низкой производительности, вероятность возникновения серьезных конфликтов.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

Риски проекта, влияющие на его ход:

- **Риски требований** — возможность изменений в требованиях к результатам.
- **Коммерческие риски** — вероятность неправильной оценки коммерческих составляющих проекта: неверной оценки рынков сбыта, времени и стоимости проекта; возможность непредвиденных расходов.
- **Управленческие риски** — вероятность выбора неправильных целей проекта, недостаточного контроля состояния проекта, возможность принятия неправильных решений и неэффективных мер.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

Риски проекта, влияющие на его ход:

- **Производственные риски** — невозможность или нерентабельность производства продукции и сбои в производстве. При производстве ПО достаточно малы, сводятся к сбоям в изготовлении коробок с продуктом.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

Риски продукта, влияющие на результаты проекта :

- **Технические риски** — возможность неуспеха в достижении запланированных показателей качества результатов проекта, вероятность вообще не получить нужный результат.
- **Эксплуатационные риски** — невозможность использования продукта или неготовность организаций- пользователей к его эксплуатации.
- **Правовые и общественные риски** — возможность возникновения патентных споров, конфликтов с коммерческими, общественными и государственными организациями по поводу самого продукта или его использования.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

Бизнес-риски :

- **Контрактные риски** — ненадежность соисполнителей, (суб)подрядчиков и поставщиков, возможность возникновения юридических претензий.
- **Инвестиционные риски** — вероятность отказа или задержек в финансировании со стороны части или всех инвесторов проекта.
- **Сбытовые риски** — возможность неполучения запланированных доходов от реализации результатов проекта, отказа пользователей от продуктов, сбоев в каналах сбыта.
- **Конъюнктурные риски** — возможность опережения проекта аналогичными проектами конкурентов, блокады ими рынка, непредвиденной конкуренции.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

- Идентификация
- Разработка стратегии преодоления рисков
- Выбор приоритетов
- Устранение или уменьшение риска

Метод расчета приоритета рисков

Вероятность (1-10) 1- маловероятно	Ущерб (1-10) 1 – слабое влияние	Стоимость устранения 1 - низкая	Расчет приоритета
Высокопри- оритетный	10	10	1 $(11-10) \times (11-10) \times 1 = 1$
Низкопри- оритетный	1	1	10 $(11-1) \times (11-1) \times 1 = 1000$

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

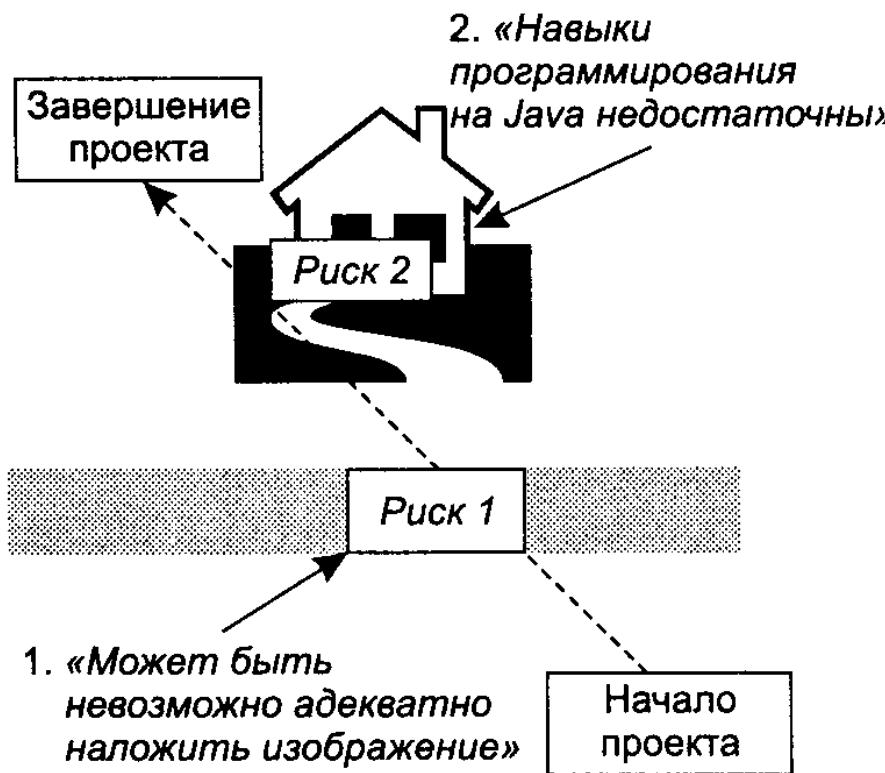
Вида стратегий преодоления рисков:

- *Стратегии предотвращения или обхода рисков.* Они направлены на снижение вероятности риска или полное избавление от него.
- *Стратегии минимизации ущерба.*
- *Планирование реакции на возникающие проблемы.*

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

Идентификация



Предупреждение



3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Ограничения

- **Ограничение (*constraint*)** — дополнительное условие, которое должен учитывать MS Project при планировании дат начала и завершения задач проекта.

Тип ограничения	Действие ограничения
Как можно раньше	Задача размещается в расписании как можно раньше. Это ограничение используется по умолчанию при планировании проекта от даты его начала
Как можно позже	Задача размещается в расписании как можно позже. Это ограничение используется по умолчанию при планировании проекта от даты его окончания

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Ограничения

Тип ограничения	Действие ограничения
Окончание не позднее заданной даты	Указанная в ограничении дата задает самую позднюю дату завершения работы. Для проекта, планируемого от даты окончания, это ограничение назначается работе, если для нее явно указать дату ее окончания.
Начало не позднее указанной даты	Заданная дата означает наиболее позднюю дату начала работы. Для проекта, планируемого от даты окончания, это ограничение назначается работе, если явно указать дату ее начала
Окончание не ранее заданной даты	Эта дата задает наиболее ранний срок завершения работы. Для проекта, планируемого от даты начала, это ограничение назначается работе, если явно указать дату ее окончания
Начало не ранее заданной даты	Эта дата означает наиболее ранний срок начала работы. Для проекта, планируемого от даты начала, это ограничение назначается работе, если явно указать дату ее начала

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Ограничения

Тип ограничения	Действие ограничения
Фиксированное начало	Работа всегда будет начинаться с указанной даты. Связи с предыдущими и последующими работами не способны изменить положение такой задачи в расписании
Фиксированное окончание	Работа всегда будет заканчиваться в указанную дату. Ее связи с другими задачами не способны изменить эту дату

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

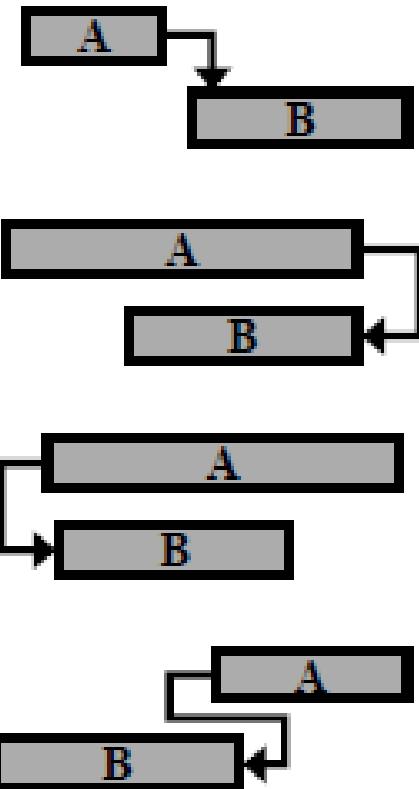
Крайний срок

- *Крайний срок (deadline)* — дата, до которой следует завершить задачу, если это условие не выполняется, MS Project выводит на экран специальный графический индикатор; в отличие от дат-ограничений, крайний срок не влияет на расписание проекта.

Последовательность операций

Используемые связи операций

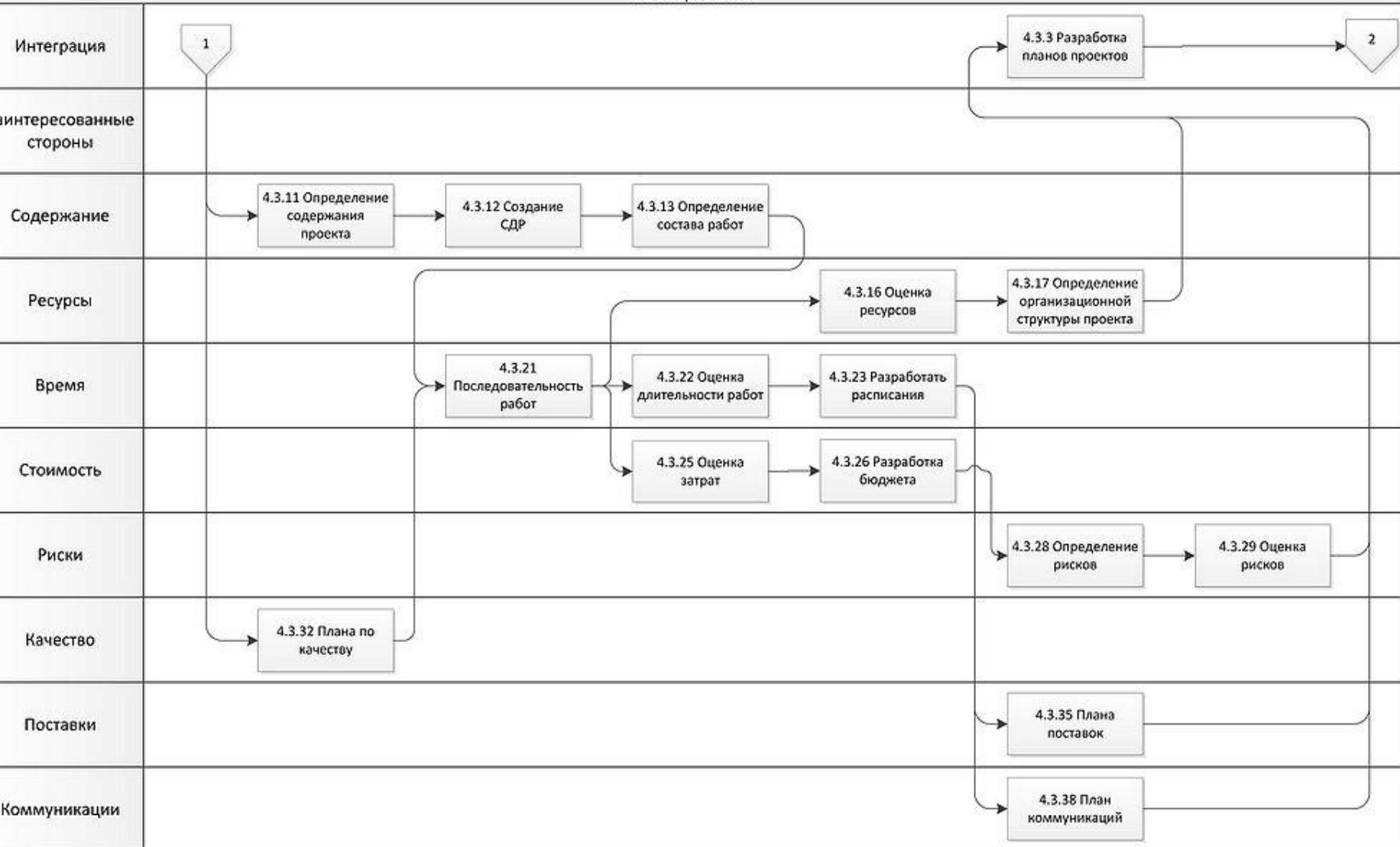
- **Финиш-старт** – последующая операция **начинается после завершения** предшествующей (тестирование после построения, стены после фундамента)
- **Финиш-финиш** – последующая операция **завершается после завершения** предшествующей (тестирование электропроводки не может завершиться до завершения прокладки)
- **Старт-старт** – последующая операция **начинается после начала** предшествующей (выравнивание бетона не может начаться до начала заливки фундамента)
- **Старт-финиш** – последующая операция **завершается после начала** предшествующей (рекламная компания завершается перед открытием нового магазина)



Процессы «Планирование»

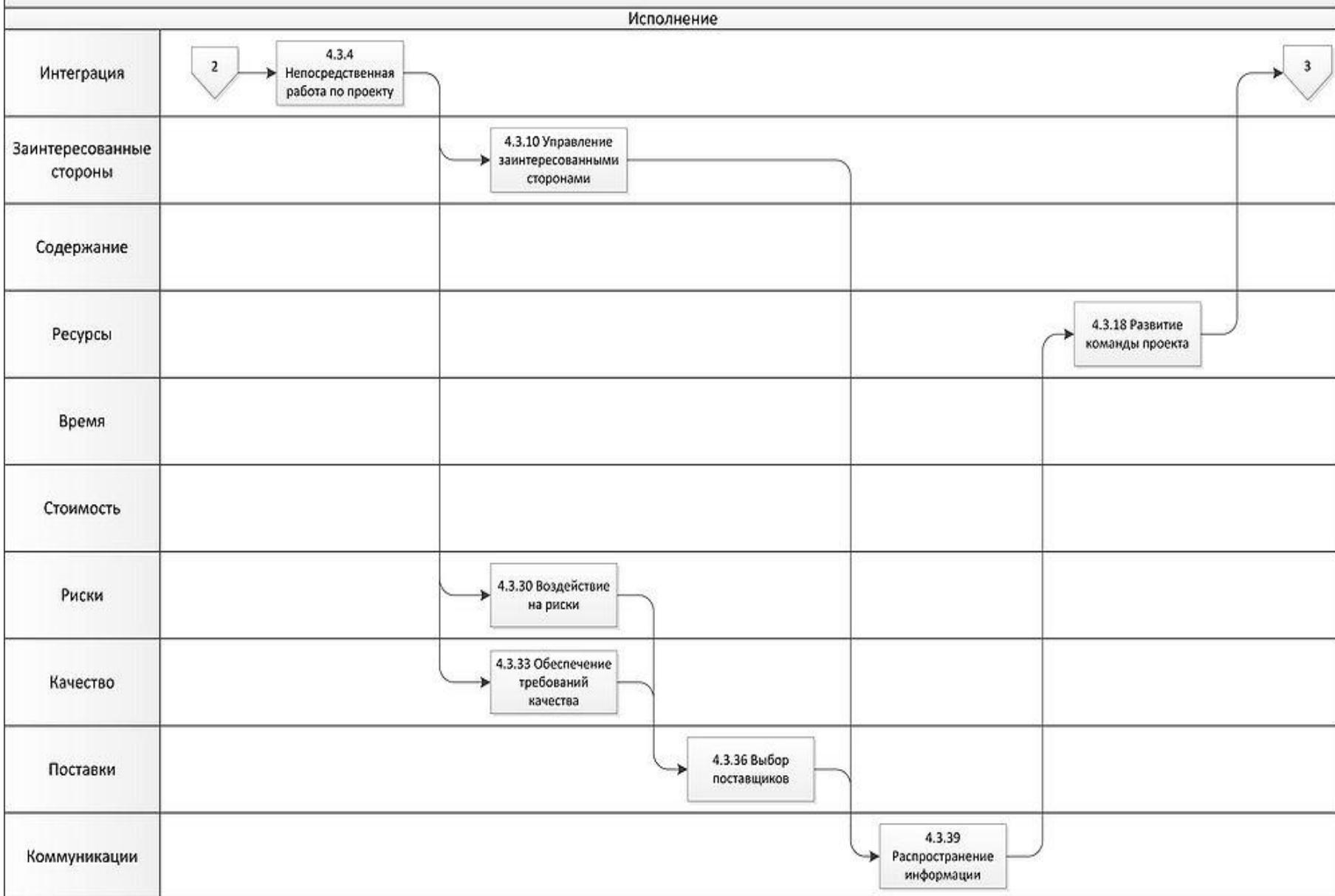
Предметные группы \ Группы процессов

Планирование



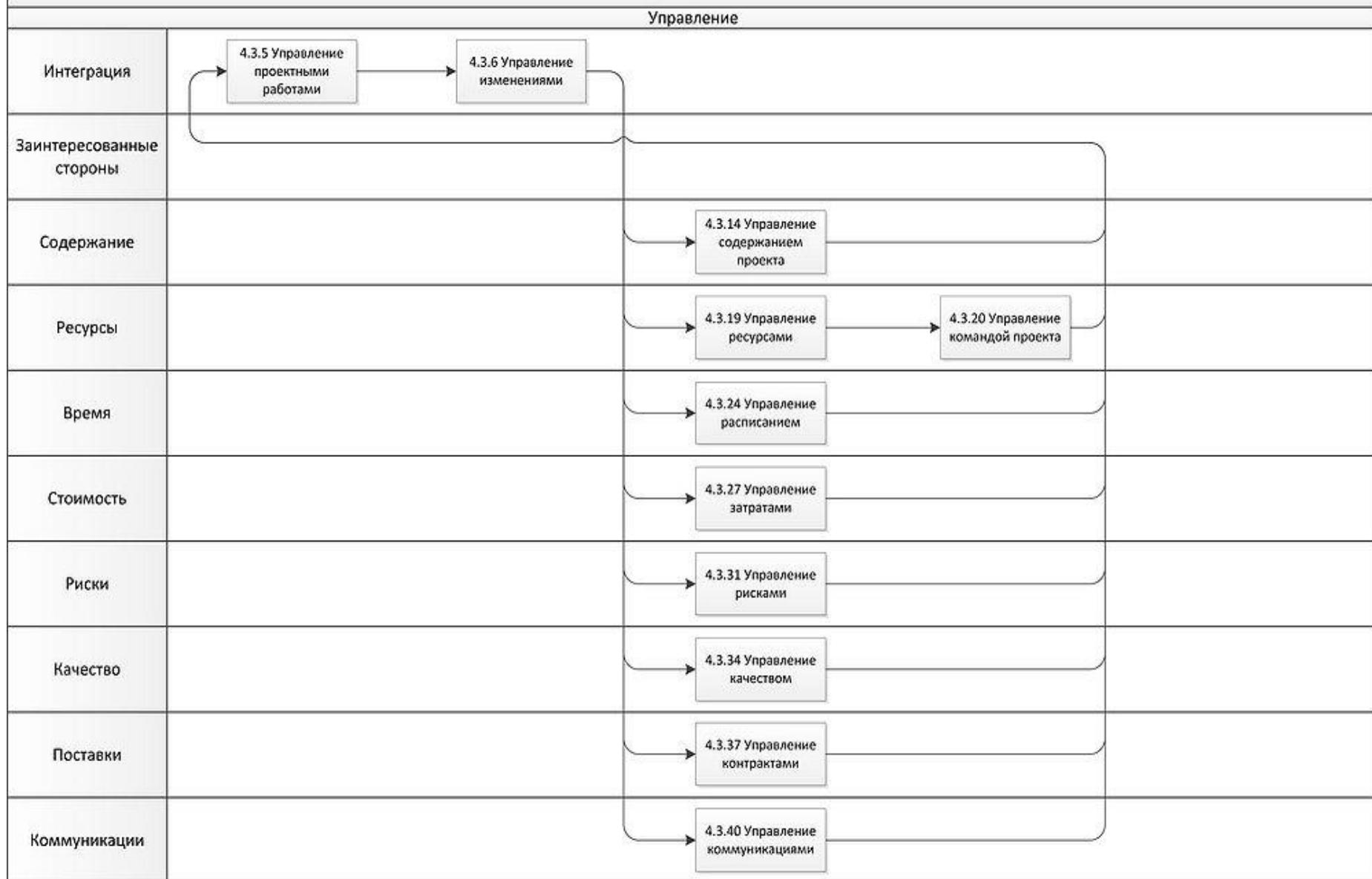
Процессы «Исполнение»

Предметные группы \ Группы процессов



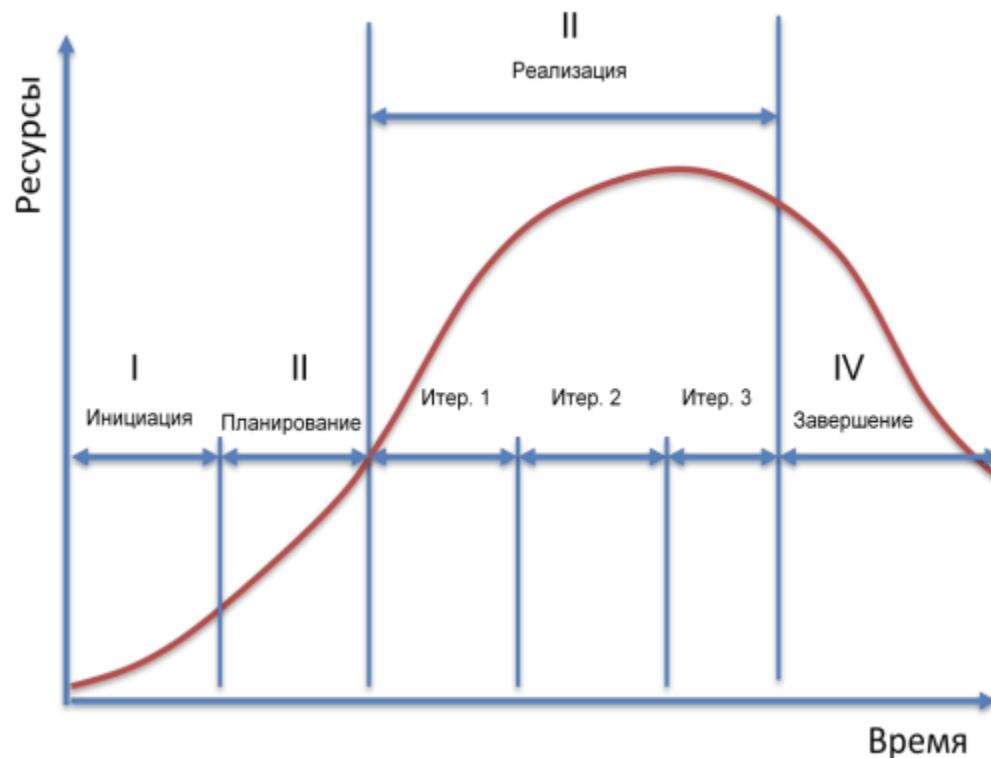
Группа процессов «Контроль»

Предметные группы \ Группы процессов



3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Распределение ресурсов по фазам проекта



Группа процессов «Завершение»

Предметные группы \ Группы процессов

Завершение	
Интеграция	<pre>graph LR; A[3] --> B["4.3.7 Закрытие отдельной фазы или проекта"]; B --> C["4.3.8 Сохранение извлеченных уроков"]; C --> D((Конец))</pre>
Заинтересованные стороны	
Содержание	
Ресурсы	
Время	
Стоимость	
Риски	
Качество	
Поставки	
Коммуникации	

Метрики ПО

- **Метрика программного обеспечения** — это мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификаций.

Необходимость метрик

- Для чего нужны метрики?
- Для того чтобы оценить программный продукт
 - Качество,
 - Надежность,
 - Цену и т.д.

Метрики ПО

- количество строк кода,
- цикломатическая сложность,
- анализ функциональных точек,
- количество ошибок на 1000 строк кода,
- степень покрытия кода тестированием,
- покрытие требований,
- количество классов и интерфейсов.

Измерения в разработке ПО

Классификация метрик по признаку их ориентации:

- размеро-ориентированные метрики, использующиеся для сбора результатов прямых измерений программного продукта и его качества, а также процесса разработки;
- функционально-ориентированные метрики, которые являются косвенными мерами, характеризующими функциональное назначение продукта и особенности его входных и выходных данных;
- человеко-ориентированные метрики, которые также являются косвенными мерами, позволяющими судить об отношении разработчиков и пользователей, об эффективности и качестве работы программного изделия, удобстве взаимодействия с ним, простоте обучения и т.д.

Измерения в разработке ПО

Размеро-ориентированные метрики базируются на числе строк кода Lines Of Code, LOC(метрика).

Регистрируются следующие показатели:

- общие затраты (в чел.-мес);
- объем программного изделия (в тысячах строк исходного кода -KLOC);
- стоимость разработки (в тыс.рублей или \$);
- объем документации (в страницах документов -СД);
- ошибки, обнаруженные в течение первого года эксплуатации (число ошибок - ЧО);
- число людей, работавших над изделием (человек);
- срок разработки (в календарных месяцах).

Измерения в разработке ПО

- Рассчитываются:

$$\text{Производительность} = \frac{\text{Длина}}{\text{Затраты}} \left[\frac{KLOC}{\text{чел. - мес}} \right],$$

$$\text{Качество} = \frac{\text{Ошибка}}{\text{Длина}} \left[\frac{\text{ЧО}}{KLOC} \right],$$

$$\text{Удельная Стоимость} = \frac{\text{Стоимость}}{\text{Длина}} \left[\frac{\text{тыс. \$}}{LOC} \right],$$

$$\text{Документированность} = \frac{\text{Страницы Документа}}{\text{Длина}} \left[\frac{СД}{KLOC} \right],$$

LOC

- Преимущества использования этого критерия
 - Простота.
- Но он имеет кучу недостатков:
 - Размер проекта в терминах LOC может быть определен только после его завершения,
 - Зависит от языка программирования,
 - Не качественная оценка работы программиста,
 - Не отражает функциональные свойства кода программы.

Измерения в разработке ПО

- **Методология оценивания функционального размера (*Functional Point*)**

Информационные характеристики:

1. Количество внешних вводов.
2. Количество внешних выводов.
3. Количество внешних запросов.
4. Количество внутренних логических файлов.
5. Количество внешних интерфейсных файлов.

Измерения в разработке ПО



Вводы, выводы и запросы относят к категории транзакция. Транзакция — это элементарный процесс, различаемый пользователем и перемещающий данные между внешней средой и программным приложением.

Измерения в разработке ПО

- ***Внутренние логические файлы (ILFs)*** — выделяемые пользователем логически связанные группы данных или блоки управляющей информации, которые поддерживаются внутри продукта.
- ***Внешние интерфейсные файлы (EIFs)*** — выделяемые пользователем логически связанные группы данных или блоки управляющей информации, на которые ссылается продукт, но которые поддерживаются вне продукта.

Измерения в разработке ПО

- **Внешний ввод** (EI, external inputs) — элементарный процесс, перемещающий данные из внешней среды в приложение.
- **Внешний вывод** (EO, external outputs) — элементарный процесс, перемещающий данные, вычисленные в приложении, во внешнюю среду. Кроме того, в этом процессе могут обновляться внутренние логические файлы.
- **Внешний запрос** (EQ, external inquiries) — элементарный процесс, работающий как с вводимыми, так и с выводимыми данными. Его результат — данные, возвращаемые из внутренних логических файлов и внешних интерфейсных файлов.

Измерения в разработке ПО

Примеры элементов данных

Информационная характеристика	Элементы данных
Внешние Вводы	Поля ввода данных, сообщения об ошибках, вычисляемые значения, кнопки
Внешние Выводы	Поля данных в отчетах, вычисляемые значения, сообщения об ошибках, заголовки столбцов, которые читаются из внутреннего файла
Внешние Запросы	Вводимые элементы: поле, используемое для поиска, щелчок мыши. Выводимые элементы — отображаемые на экране поля

Измерения в разработке ПО

Каждой характеристике назначается низкий, средний или высокий ранг, затем формируется числовая оценка ранга.

Расчет количества функциональных указателей FP (Function Points)

Имя характеристики	Ранг, сложность, количество			
	Низкий	Средний	Высокий	Итого
Внешние вводы	? x 3 =	? x 4 =	? x 6 =	= ?
Внешние выводы	? x 4 =	? x 5 =	? x 7 =	= ?
Внешние запросы	? x 3 =	? x 4 =	? x 6 =	= ?
Внутренние логические файлы	? x 7 =	? x 10 =	? x 15 =	= ?
Внешние интерфейсные	? x 5 =	? x 7 =	? x 10 =	= ?

Измерения в разработке ПО

Расчет количества функциональных указателей FP (Function Points)

$$FP = \text{Общее количество} \times (0,65 + 0,01 \times \sum_{i=1}^{14} F_i),$$

где F_i — коэффициенты регулировки сложности (14).

Каждый коэффициент может принимать следующие значения: 0 - нет влияния, 1 - случайное, 2 - небольшое, 3 - среднее, 4 - важное, 5 - основное.

Измерения в разработке ПО

Расчет количества функциональных указателей FP (Function Points)

$$\text{Производительность} = \frac{\text{Функций Указатель}}{\text{Затраты}} \left[\frac{FP}{\text{чел. - мес}} \right],$$

$$\text{Качество} = \frac{\text{Ошибки}}{\text{Функций Указатель}} \left[\frac{ЧО}{FP} \right],$$

$$\text{Удельная Стоимость} = \frac{\text{Стоимость}}{\text{Функций Указатель}} \left[\frac{\text{тыс. \$}}{FP} \right],$$

$$\text{Документированность} = \frac{\text{Страниц Документа}}{\text{Функций Указатель}} \left[\frac{СД}{FP} \right],$$

Измерения в разработке ПО

Пересчет FP-оценок в LOC-оценки

Язык программирования	Количество операторов на один FP
Ассемблер	320
С	128
Фортран	106
Паскаль	90
C++	64
Java	53
Ada95	49
Visual Basic	32
Visual C+	34
Delphi(Pascal)	29
Smalltalk	22
HTML3	15
LISP	64
Prolog	64

Измерения в разработке ПО

Для оценки трудоемкости и времени разработки может использоваться модель ***COCOMO II***
(Cost Constructive Model).

$$PM = A \times SIZE^E \times \prod_{i=1}^n EM_i$$

SIZE — размер продукта в KLOC;

$$A = 2,94$$

$$E = B + 0,01 \times \sum_{j=1}^5 SF_j$$

$$B = 0,91$$

EM_i — множители трудоемкости;

SF_j — факторы масштаба

n=7 — для предварительной оценки

n=17 — для детальной оценки

Измерения в разработке ПО

Cost Constructive Model

Состав коэффициентов EM_i :

- 1) требуемая надежность (0.82-1.26);
- 2) размер базы данных (0.90-1.28);
- 3) сложность продукта (0.73-1.74);
- 4) требуемый уровень повторного использования (0.95-1.24);
- 5) соответствие документации требованиям жц (0.81-1.23)
- 6) ограничение времени выполнения (1.00-1.63);
- 7) ограничение по объему основной памяти (1.00-1.46);

Измерения в разработке ПО *Cost Constructive Model*

Состав коэффициентов EM_i :

- 8) изменчивость платформы (0.87-1.30);
- 9) способности аналитика (0.42-0.71);
- 10) способности программиста (0.34-0.76);
- 11) знание приложений (0.22-0.81);
- 12) знание платформы (0.19-0.85);
- 13) преемственность персонала (0.29-0.81);
- 14) знание языка/инструментальных средств (0.20-0.84);

Измерения в разработке ПО *Cost Constructive Model*

Состав коэффициентов EM_i :

- 15) использование инструментальных средств
(1.17-0.78);
- 16) требуемые сроки разработки (1.43-1.00);
- 17) рассредоточенность команды разработчиков
(1.22-0.80).

Измерения в разработке ПО ***Cost Constructive Model***

Факторы SF_j :

- 1) наличие прецедентов у приложения: степень опытности организаций-разработчика в данной области;
- 2) гибкость процесса: степень строгости контракта, порядок его выполнения;
- 3) разрешение рисков, присущих архитектуре;
- 4) сплоченность команды;
- 5) зрелость процесса: уровень зрелости организации –разработчика.

Измерения в разработке ПО

Измерения при разработке ПО необходимы для того, чтобы:

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

Функциональные возможности MS Project



- Создание графика работ
- Создание перечня ресурсов и оптимизация их загрузки
- Расчет стоимости использования ресурсов
- Построение бюджета проекта
- Анализ проекта
- Контроль и управление проектом
- Создание стандартных отчетов
- Управление группой проектов
- Организация работ в рабочей группе
- Интеграция с Интернет
- Обмен данными с приложениями MS Office

Составляющие проектного плана

- ▶ **Задача (Task)** - работа, осуществляемая в рамках проекта для достижения определенного результата.
- **Фаза проекта или суммарная задача (Summary Task)** состоит из одной или нескольких задач, в результате выполнения которых достигается один или несколько основных результатов проекта.
- ▶ Задачи, в результате исполнения которых достигаются промежуточные цели, называются **завершающими задачами или вехами (Milestone)**.

Составляющие проектного плана

- ▶ *Длительность задачи (Duration)* — это период рабочего времени, который необходим для выполнения задачи.
- ▶ *Трудозатраты (Work)* — время, затраченное сотрудниками на получение результата.
- *Зависимость или связь (Link)* — логическая взаимосвязь между задачами проекта, определяющая порядок их выполнения.

Составляющие проектного плана

- *Ресурсы* - сотрудники и оборудование, необходимые для выполнения проектных задач. Характеристика ресурсов - стоимость (Cost (Затраты)) их использования в проекте.
- *Назначение* - связь определенной задачи и ресурсов необходимых для ее выполнения. Для назначений под трудозатратами понимается объем работ, назначенный ресурсу по конкретной задаче.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ



1.1.2. Исторический и современный взгляд на разработку ПО Разработка ПО. 1950-1980

Этап вплоть до конца 1970х годов можно считать "темными веками" индустрии разработки ПО. Особенностями данного периода ее развития являются:

Общая неразвитость индустрии. Это вызвано несовершенством как технических средств, так и отсутствием теоретического базиса.

Специфическое ПО. ПО является, по сути дела, штучным продуктом, в основном используемым там же, где и велась его разработка, причем основная масса ПО – это научные и инженерные задачи.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Поначалу стандартизации вообще не придавалось никакого значения: написание программ было скорее творчеством, чем четко определенным процессом, и подчинялось **принципу "Кодирование–устранение ошибок"**. Этот подход может быть представлен в виде следующих активностей:

- ✓ Постановка задачи.
- ✓ Ее выполнение до получения требуемого результата.
- ✓ Если результат не удовлетворяет, возврат к первому шагу.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Понятно, что при такой постановке работ получить ответ на вопрос "Сколько нам понадобится времени для создания того-то и того-то" не представляется возможным. Однако, уже к середине 1970х годов появляются внутрикорпоративные стандарты разработки, соответствующие водопадной модели.

Также в 1976 году выходит ставшая классикой **книга Брукса** "**Мифический человеко-месяц**", которая не утратила своей актуальности и поныне, и всячески рекомендуется к прочтению.

Разработка ПО. 1980-1996

Данный период отрасли можно охарактеризовать как "осознание того, что так жить дальше нельзя". Аппаратные средства стали доступными как организациям, так и индивидуальным пользователям, что вызвало грандиозное увеличение объема рынка для программного обеспечения. И можно по праву назвать это время эпохой "триумфального шествия бизнес-приложений". ПО стали "потреблять" не только в местах его разработки, что вызывало, ужесточение требований как к самому ПО, так и усложнению процесса разработки.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Чаще всего часто заказчик располагался за сотни миль от места воплощения его идей в жизнь. Более того, разработчики перестали являться специалистами в предметной области заказчика (как это было ранее). Вместе с повышенной сложностью ПО и увеличившимися трудозатратами на его создание эти факторы привели к тому, что на выходе, **после водопадной модели, чаще всего заказчик получал совсем не то, что ему нужно, и продукт отправлялся в мусорную корзину, а вместе с ним – и миллионы долларов.**

Выходом из подобной ситуации стала **инкрементальная и спиральная модели.**

Одновременно делается попытка приспособить существующие в промышленности стандарты качества для ИТ-индустрии. На западе – это стандарты ISO, в СССР – ГОСТы.

Разработка ПО. Современность

Современный этап разработки ПО кратко возможно определить как "быстрее, выше, сильнее". Сложность приложений и их объем повысились еще на несколько порядков, так же, как и стоимость разработки. А одной из основных тенденций стала не просто разработка качественного продукта, а возврат инвестиций от него.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Более того, практика показала ограниченность применявшимися ранее инкрементальной и спиральной моделей, и на смену им появилась и была почти повсеместно **принята итеративная модель разработки ПО**. Стоит отметить, что большинство успешных проектов было создано именно на ее основе.

Итеративная модель была впервые предложена широким массам ИТ-специалистов компанией Rational (сейчас IBM), и является основой ее методологии RUP. Также она лежит в основе MSF (Microsoft Solutions Framework) и борландовской ALM (Application Lifecycle Management).

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В связи с тем, что разработка ПО постоянно усложняется во всех аспектах, обнаружилась потребность давать четкий ответ на вопрос **"Может ли эта организация разработать требуемый продукт?"** И, как развитие вопроса, возникла необходимость в наборе свойств, признаков, критериев, позволяющих количественно оценить степень зрелости организации, вероятность ее успеха на ниве создания ПО, так сказать. Отсюда берет корни модель СММ (в дальнейшем развивающаяся в СММІ), **Capability Maturity Model**.

Вместе с тем продолжается совершенствование стандартов разработки ПО, организации самого процесса разработки, что нашло отражение в **стандартах ISO 90003:2004 и ISO/IEC 15504**.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ



1.1.2. Типичная схема разработки ПО

Жизненный цикл программного обеспечения (ПО) — период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации [Стандарт IEEE Std 610.12, Глоссарий]

Стандарты жизненного цикла ПО

- ✓ ГОСТ 34.601-90
- ✓ ISO/IEC 12207:1995 (российский аналог — ГОСТ Р ИСО/МЭК 12207-99)

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Стандарт ГОСТ 34.601-90 предусматривает следующие стадии и **этапы создания автоматизированной системы (АС)**:

1. Формирование требований к АС

- Обследование объекта и обоснование необходимости создания АС.
- Формирование требований пользователя к АС.
- Оформление отчета о выполнении работ и заявки на разработку АС.

2. Разработка концепции АС

- Изучение объекта.
- Проведение необходимых научно-исследовательских работ.
- Разработка вариантов концепции АС и выбор варианта концепции АС, удовлетворяющего требованиям пользователей.
- Оформление отчета о проделанной работе.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3. Техническое задание

- Разработка и утверждение технического задания на создание АС.

4. Эскизный проект

- Разработка предварительных проектных решений по системе и ее частям.
- Разработка документации на АС и ее части.

5. Технический проект

- Разработка проектных решений по системе и ее частям.
- Разработка документации на АС и ее части.
- Разработка и оформление документации на поставку комплектующих изделий.
- Разработка заданий на проектирование в смежных частях проекта.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

6. Рабочая документация

- Разработка рабочей документации на АС и ее части.
- Разработка и адаптация программ.

7. Ввод в действие

- Подготовка объекта автоматизации.
- Подготовка персонала.
- Комплектация АС поставляемыми изделиями (программными и техническими средствами, программно-техническими комплексами, информационными изделиями).
- Строительно-монтажные работы.
- Пусконаладочные работы.
- Проведение предварительных испытаний.
- Проведение опытной эксплуатации.
- Проведение приемочных испытаний.

8. Сопровождение АС.

- Выполнение работ в соответствии с гарантийными обязательствами.
- Послегарантийное обслуживание.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Стандартная последовательность шагов разработки ПО.

1. Понять природу и сферу применения предлагаемого продукта.
2. Выбрать процесс разработки и создать план.
3. Собрать требования.
4. Спроектировать и собрать продукт.
5. Выполнить тестирование продукта.
6. Выпустить продукт и обеспечить его сопровождение.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Разновидности *моделей* процесса разработки ПО:

- водопадная;
- спиральная (итерационная);
- инкрементная;
- макетирование;
- объектно-ориентированное программирование;
- компонентный подход (COM, CORBA);
- CASE-технологии;
- RAD;
- RUP;
- гибкие (agile) технологии: экстремальное программирование (XP), Scrum, TDD, FDD.

1.2. Традиционные технологии разработки ПО

1.2.1. Водопадная (waterfall) модель

Водопадная или каскадная модель жизненного цикла

(англ. *waterfall model*) была предложена в 1970 г. Уинстоном Ройсом. Она предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе. Разработка рассматривается как последовательность этапов, причем переход на следующий, иерархически нижний этап происходит только после полного завершения работ на текущем этапе (рис. 1.1).

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

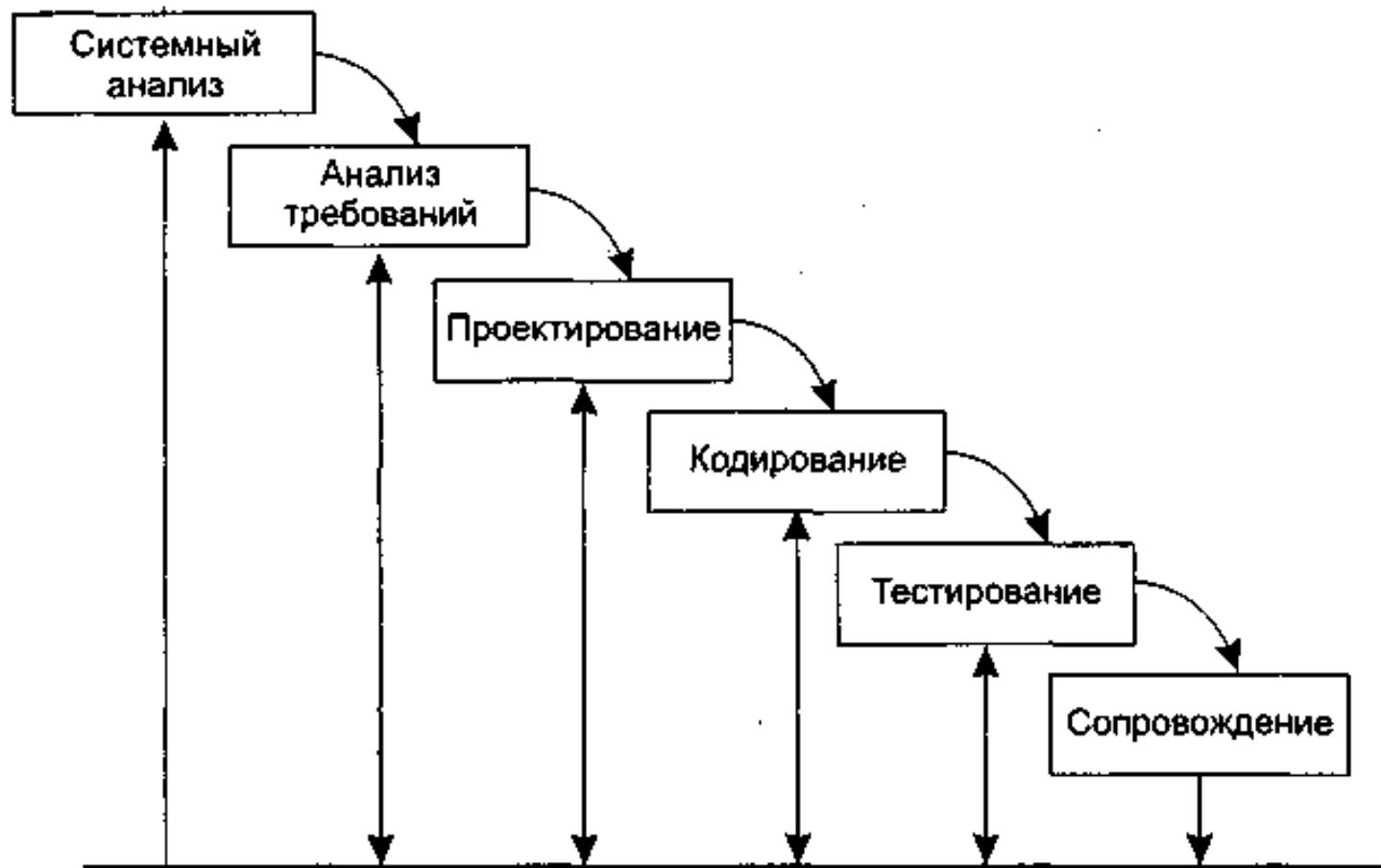


Рис. 1.1. Классический жизненный цикл разработки ПО

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Системный анализ задает роль каждого элемента в компьютерной системе, взаимодействие элементов друг с другом.

Анализ требований относится к программному элементу — программному обеспечению.

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.

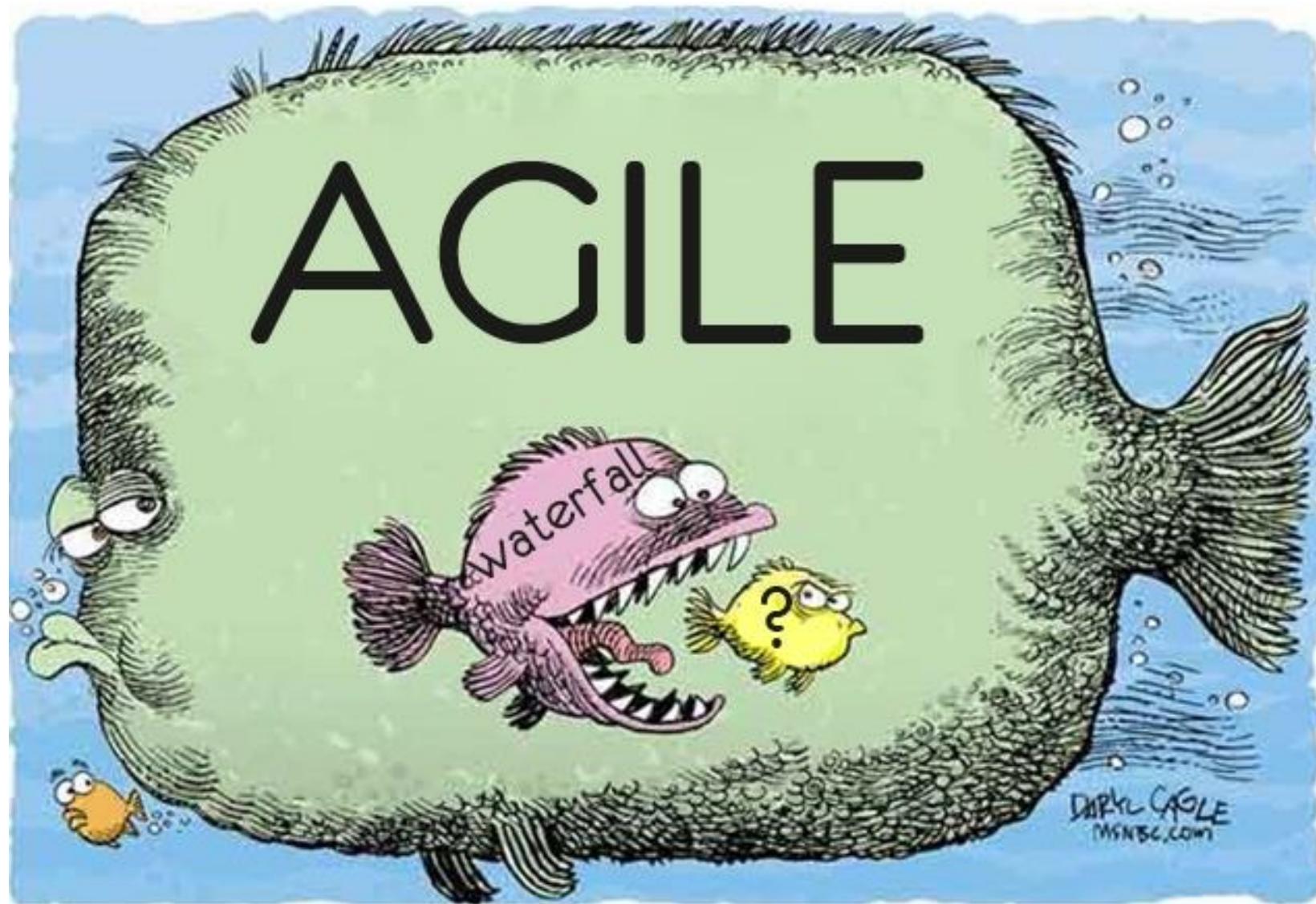
ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Достоинства водопадной модели:

- ✓ дает план и временной график по всем этапам проекта;
- ✓ упорядочивает ход конструирования;
- ✓ полная и согласованная документация на каждом этапе;
- ✓ легко определить сроки и затраты на проект.

Недостатки водопадной модели:

- ✓ 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.

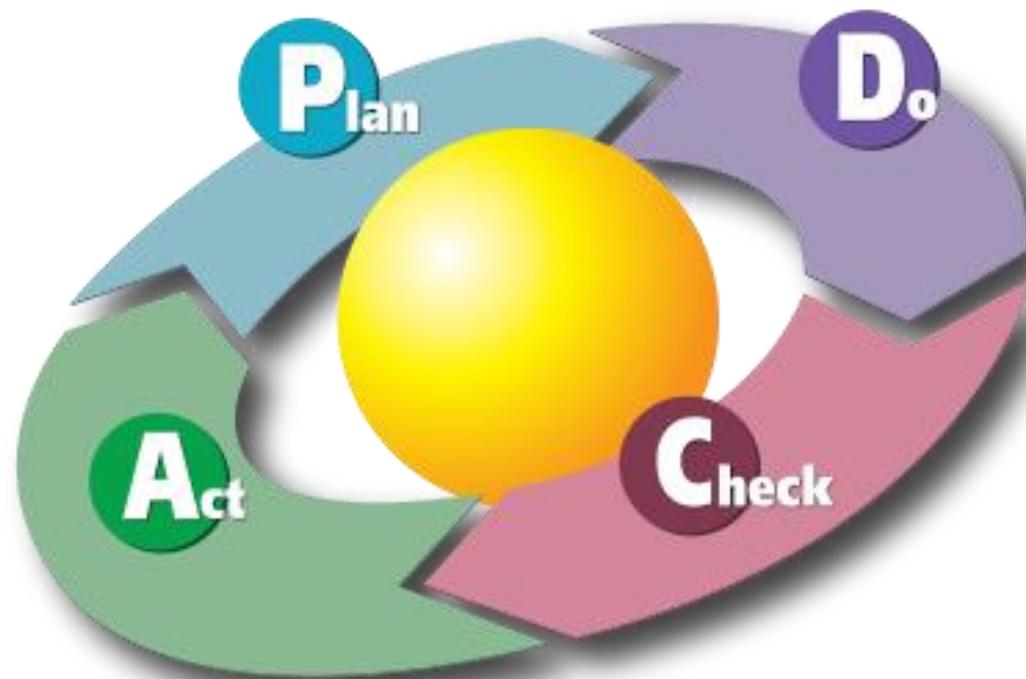


1.2.2. Спиральная (spiral) модель

Спиральная модель (англ. *spiral mode*) была разработана в середине 1980-х годов Барри Боэмом. Она основана на классическом цикле Деминга PDCA (plan-do-check-act). При использовании этой модели ПО создается в несколько итераций (витков спирали) методом прототипирования.

Каждая итерация соответствует созданию фрагмента или версии ПО, на ней уточняются цели и характеристики проекта, оценивается качество полученных результатов и планируются работы следующей итерации.

PDCA («Plan-Do-Check-Act») циклически повторяющийся процесс принятия решения, используемый в управлении качеством. Также известен как **Deming Cycle**, **Shewhart cycle**, **Deming Wheel**, или **Plan-Do-Study-Act**.



ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методология PDCA представляет собой простейший алгоритм действий руководителя по управлению процессом и достижению его целей. Цикл управления начинается с планирования.

Планирование

установление целей и процессов, необходимых для достижения целей, планирование работ по достижению целей процесса и удовлетворения потребителя, планирование выделения и распределения необходимых ресурсов.

Выполнение

выполнение запланированных работ.

Проверка

сбор информации и контроль результата на основе ключевых показателей эффективности (KPI), получившегося в ходе выполнения процесса, выявление и анализ отклонений, установление причин отклонений.

Воздействие (управление, корректировка)

принятие мер по устранению причин отклонений от запланированного результата, изменения в планировании и распределении ресурсов.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ



Рис. 1.2. 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

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

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.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

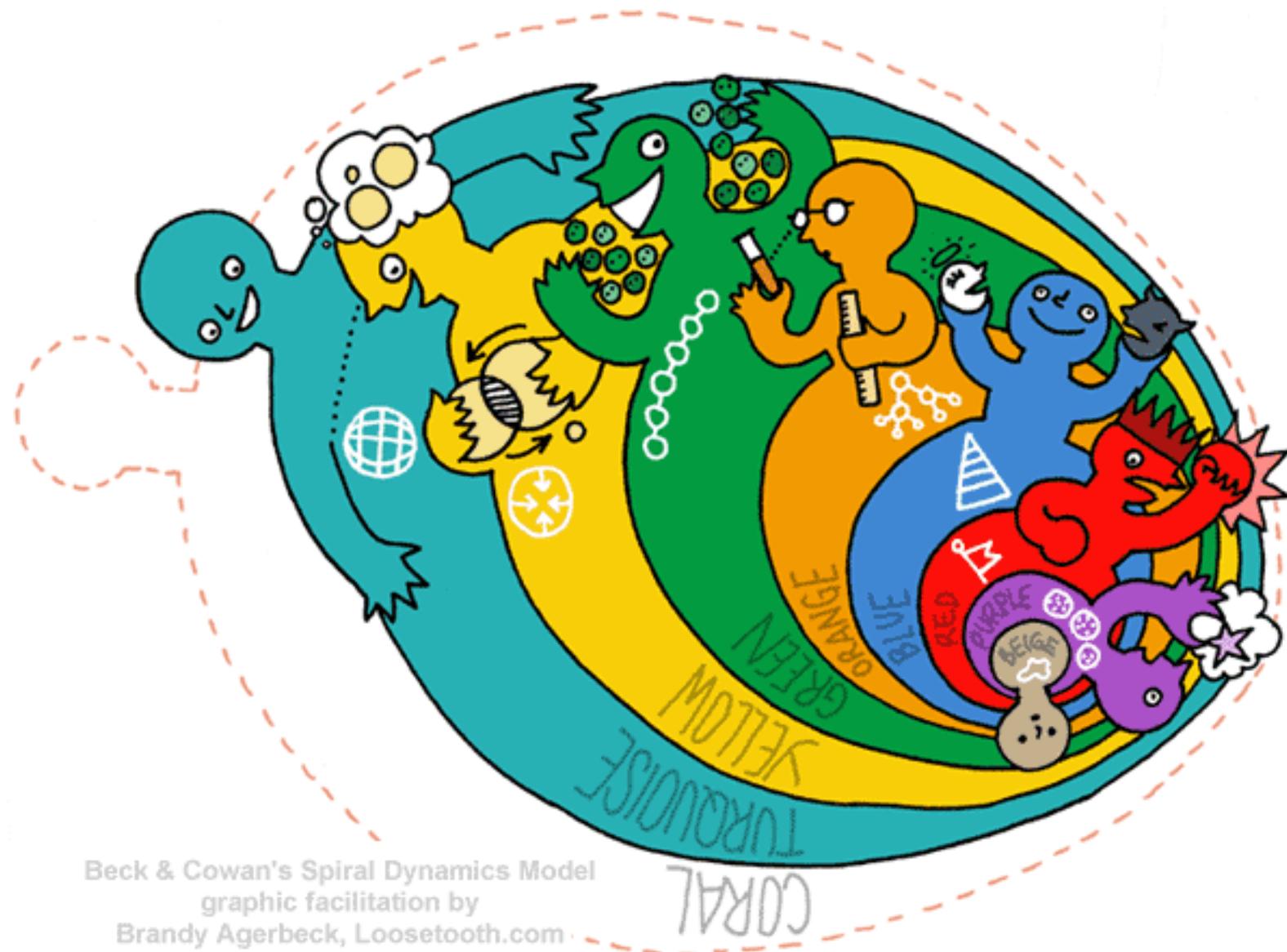
Достоинства спиральной модели:

- 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.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ



ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.2.3. Инкрементная модель

Модель итеративной и инкрементальной разработки

(англ. *iterative and incremental development, IID*), получившей также от Т. Гилба в 70-е гг. название эволюционной модели.

Модель IID предполагает разбиение жизненного цикла проекта на последовательность итераций, каждая из которых напоминает «мини-проект», включая все процессы разработки в применении к созданию меньших фрагментов функциональности, по сравнению с проектом в целом.

Цель каждой итерации — получение работающей версии программной системы, включающей функциональность, определённую интегрированным содержанием всех предыдущих и текущей итерации. Результат финальной итерации содержит всю требуемую функциональность продукта. Таким образом, с завершением каждой итерации продукт получает приращение — **инкремент** — к его возможностям, которые, следовательно, развиваются **эволюционно**. Итеративность, инкрементальность и эволюционность в данном случае есть выражение одного и то же смысла разными словами со слегка разных точек зрения.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

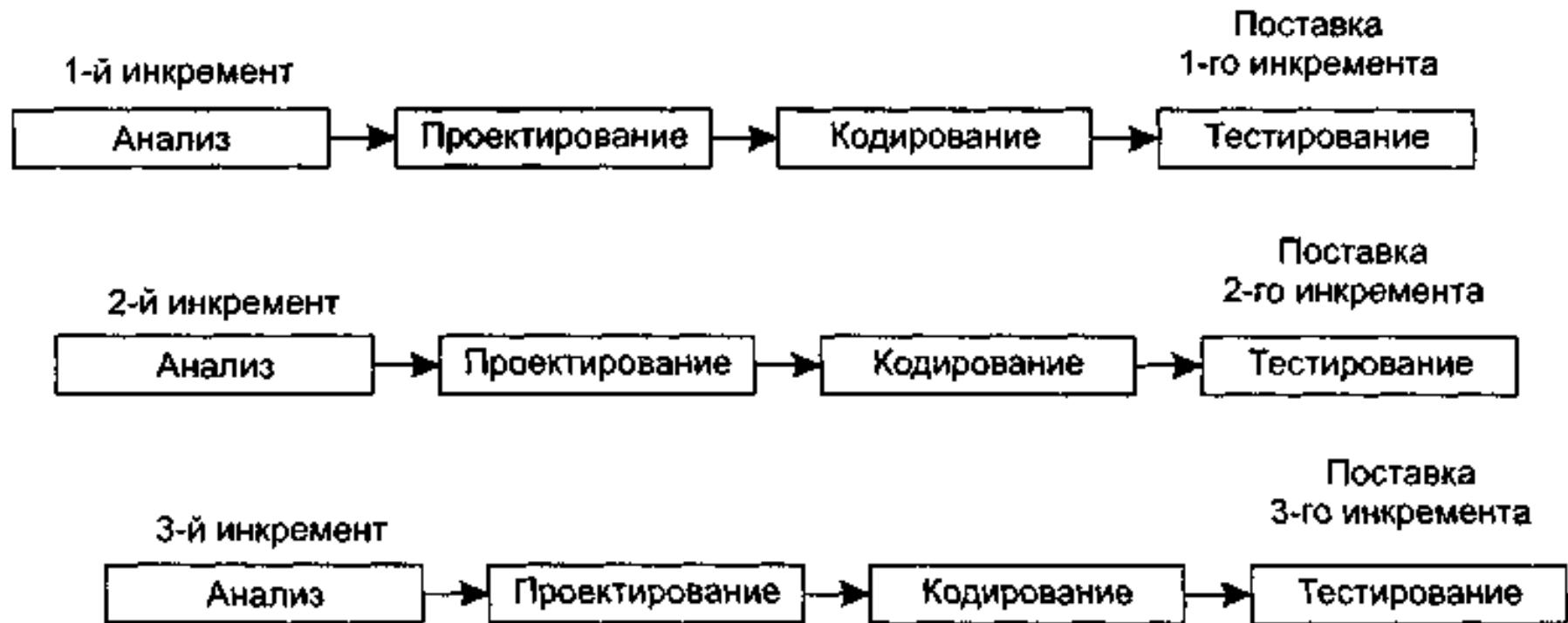


Рис. 1.3. Инкрементная модель

Достоинства итеративной модели:

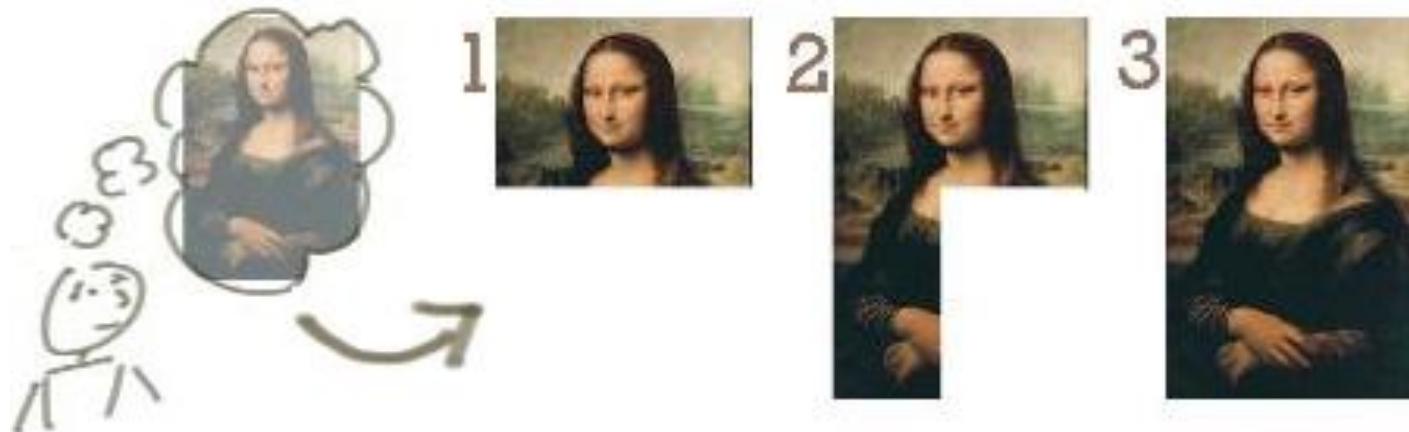
По выражению Т. Гилба, «эволюция — прием, предназначенный для создания видимости стабильности. Шансы успешного создания сложной системы будут максимальными, если она реализуется в серии небольших шагов и если каждый шаг заключает в себе четко определённый успех, а также возможность «отката» к предыдущему успешному этапу в случае неудачи.

Перед тем, как пустить в дело все ресурсы, предназначенные для создания системы, разработчик имеет возможность получать из реального мира сигналы обратной связи и исправлять возможные ошибки в проекте» [Ларман К. Итеративная и инкрементальная разработка: краткая история / К. Ларман, В. Базили // Открытые системы. — 2003.— N 9]

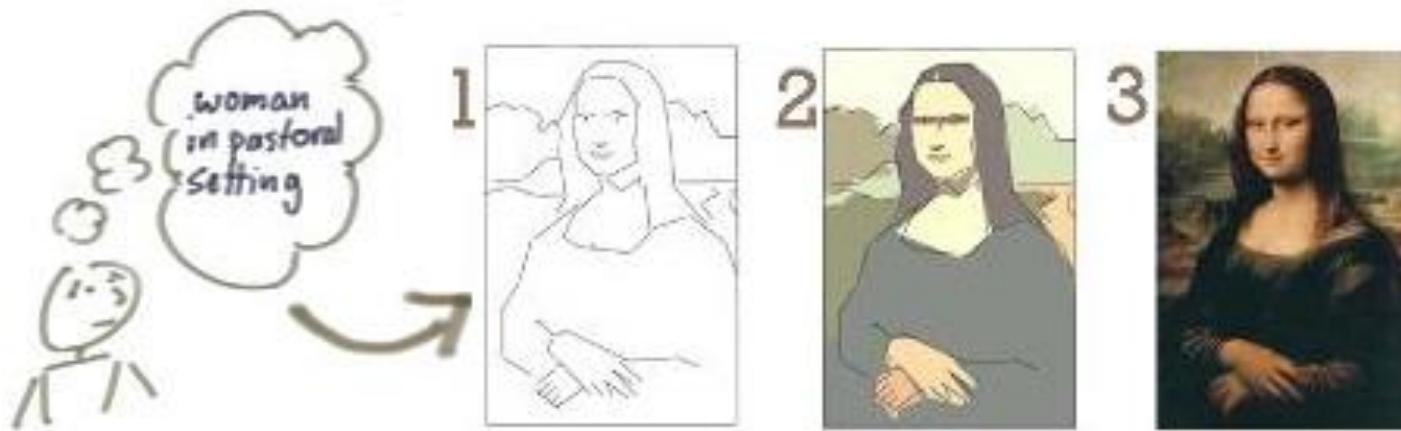
Недостатки итеративной модели:

- 1) целостное понимание возможностей и ограничений проекта очень долгое время отсутствует;
- 2) при итерациях приходится отбрасывать часть сделанной ранее работы;
- 3) добросовестность специалистов при выполнении работ всё же снижается, что психологически объяснимо, ведь над ними постоянно довлеет ощущение, что «всё равно всё можно будет переделать и улучшить позже»;
Различные варианты итерационного подхода реализованы в большинстве современных методологий разработки (RUP, MSF, XP).

Incremental



Iterative



1.2.4. Макетирование или прототипирование (prototyping)

Прототипирование программного обеспечения (от англ. prototyping) — этап разработки программного обеспечения (ПО), процесс создания **прототипа программы** — макета (черновой, пробной версии) программы, обычно — с целью проверки пригодности предлагаемых для применения концепций, архитектурных и/или технологических решений, а также для представления программы заказчику на ранних стадиях процесса разработки.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

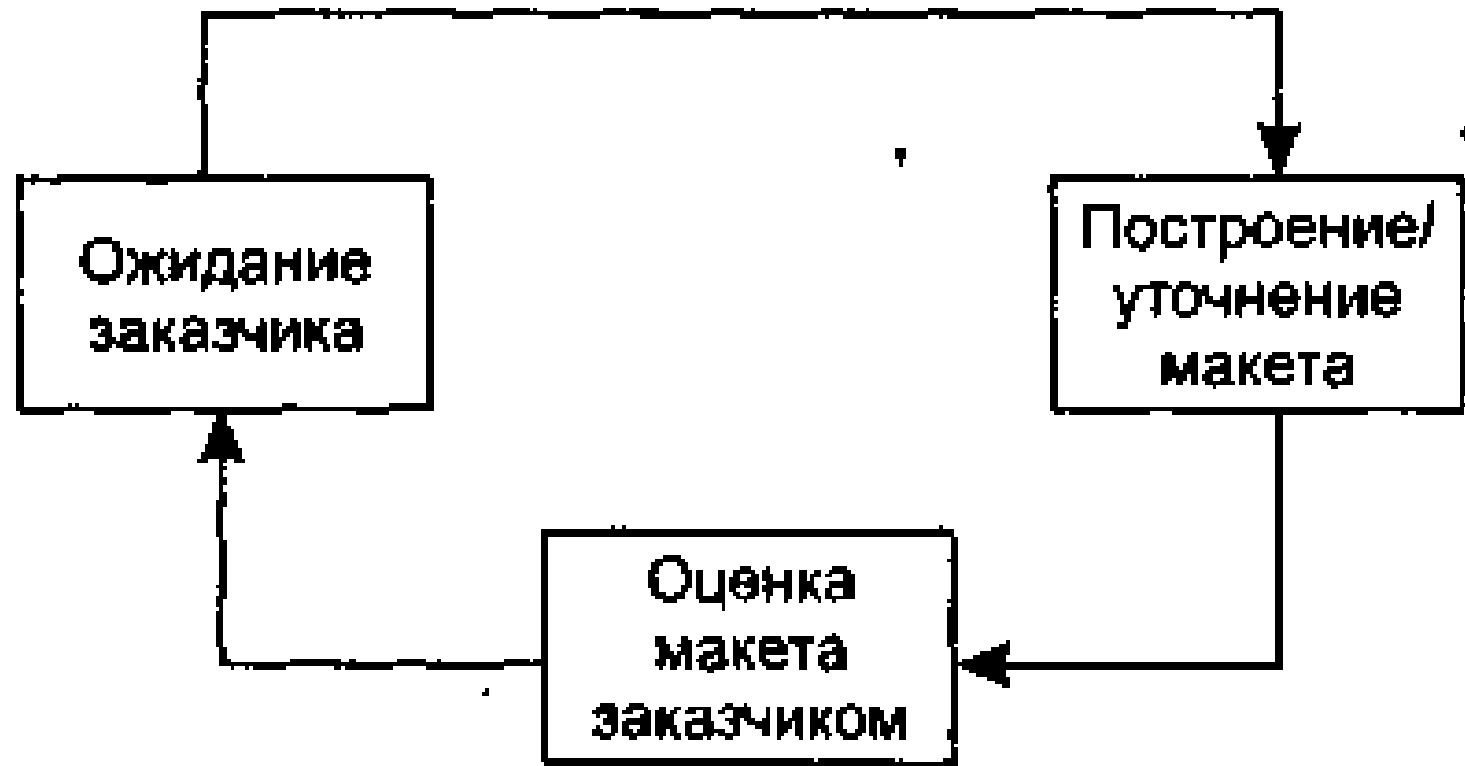


Рис. 1.4. Макетирование

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

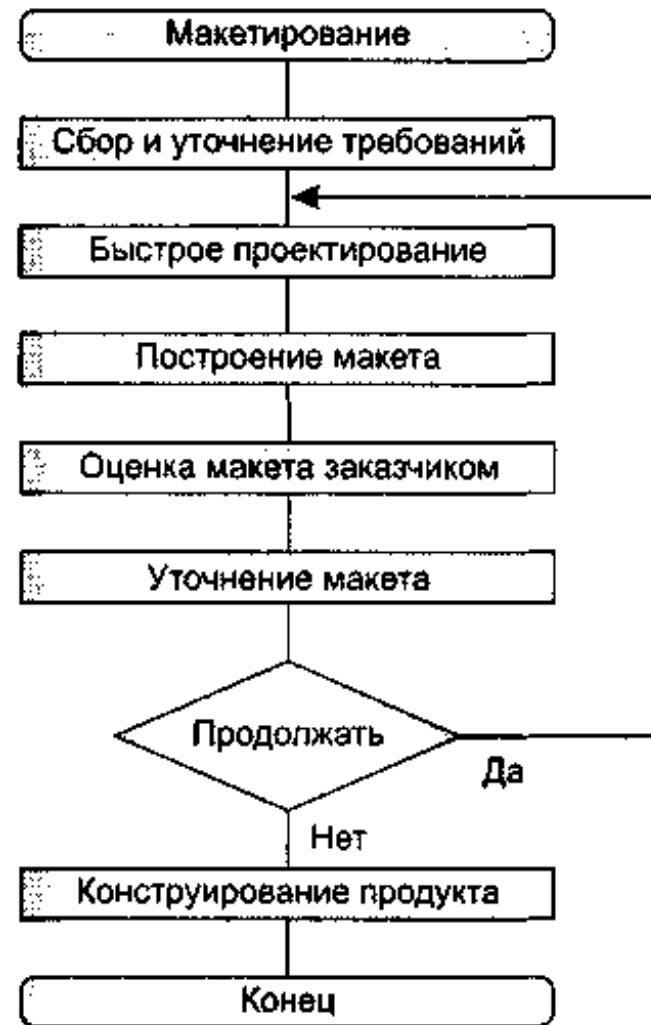


Рис. 1.5. Последовательность действий при макетировании

Быстрое прототипирование

При *быстром прототипировании* (англ. *Rapid prototyping* или *throwaway prototyping*) предполагается, что мы создаем макет, который на каком-то этапе будет оставлен («выброшен») и не станет частью готовой системы.

Эволюционное прототипирование

Эволюционное прототипирование (англ. *evolutionary prototyping*) ставит своей целью последовательно создавать макеты системы, которые будут все ближе и ближе к реальному продукту.

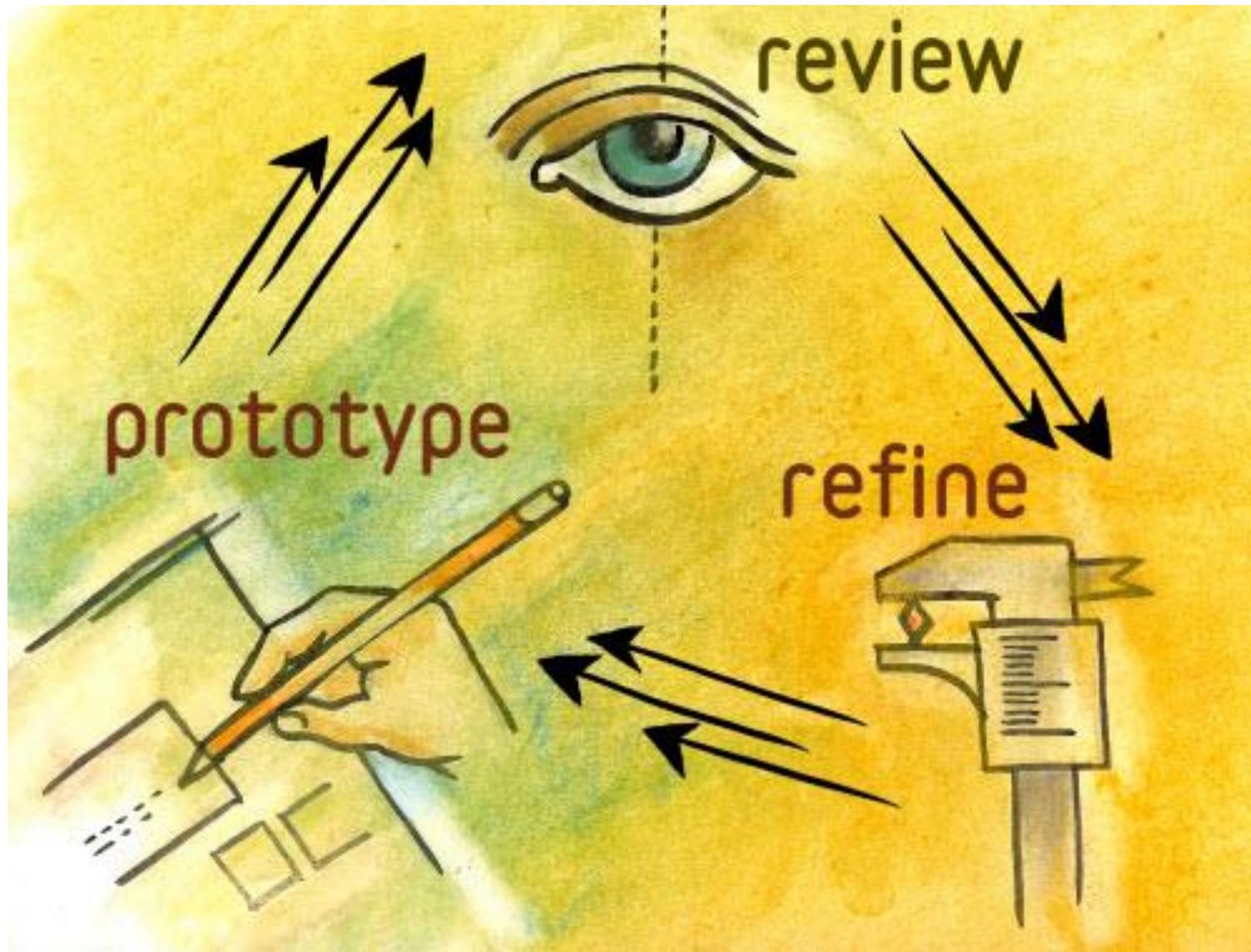
Достоинство прототипирования:

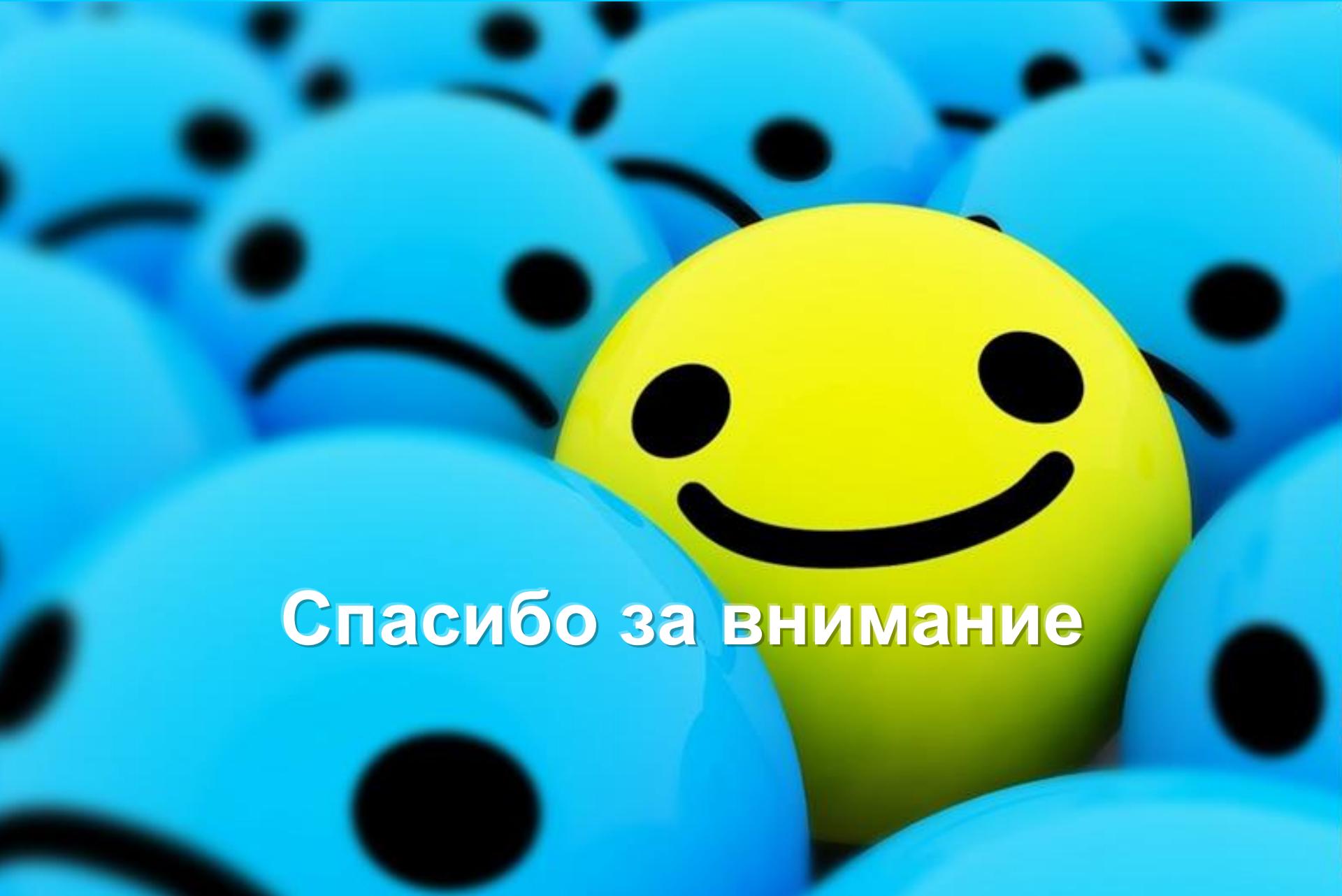
- 1) обеспечивает определение полных требований к ПО.
- 2) уменьшение времени, стоимости, рисков;
- 3) вовлечение пользователя в процесс разработки.

Недостатки прототипирования :

- 1) заказчик может принять макет за продукт;
- 2) разработчик может принять макет за продукт;
недостаточный анализ;
- 3) смешение прототипа и готовой системы в представлении
пользователей;
- 4) чрезмерное время на создание прототипа.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ





Спасибо за внимание

Инкрементная модель экстремального программирования: недостатки

- успех проекта зависит от вовлеченности заказчика и от уровня программистов, методология работает только с senior специалистами
- трудно предугадать затраты времени на проект, т.к. в начале никто не знает полного списка требований
- менеджмент негативно относится к парному программированию (почему нужно оплачивать двух программистов вместо одного?)
- регулярные встречи с программистами дорого обходятся заказчикам
- из-за недостатка структуры и документации не подходит для крупных проектов
- т.к. гибкие методологии функционально-ориентированные, нефункциональные требования к качеству продукта сложно описать в виде пользовательских историй

ОРГАНИЗАЦИЯ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Лекция 2

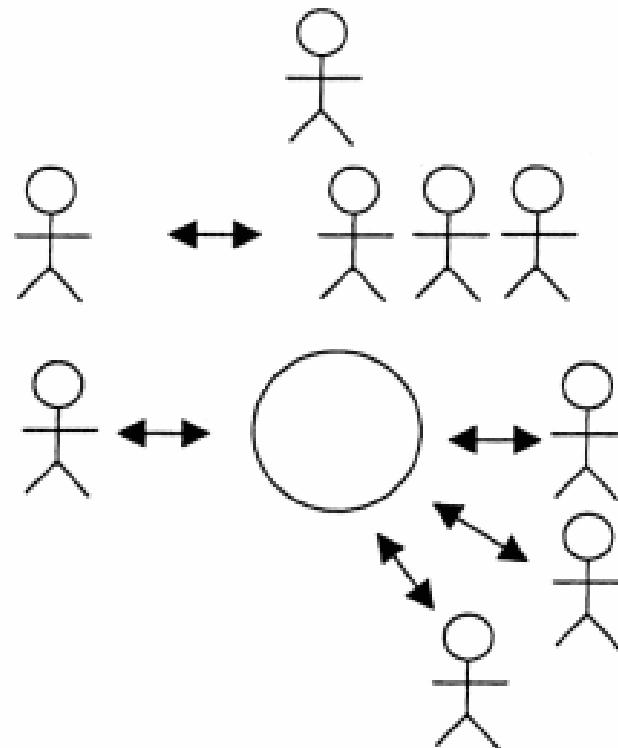
Управление разработкой программного
обеспечения

ОРГАНИЗАЦИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1. Группа проекта и роли участников
2. Документация процесса разработки программного обеспечения.
3. Управление проектом разработки ПО

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Триада разработок в зависимости от количества участников



1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Авторская разработка

Жизненный цикл разработки поддерживается одним единственным человеком (70-80-е годы ХХ века).

Применение в области наукоемких приложений:

- необходимость многолетнего изучения предметной области,
- практически полное отсутствие начального финансирования проекта,
- малая рентабельность, определяемая узким кругом пользователей.

Объем программного продукта, выполненного методом авторской разработки, в 5-20 раз меньше по сравнению с индустриальными аналогами.

Авторская разработка предполагает достижение профессионального успеха и известности .

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Авторская разработка

Иногда авторская разработка может выигрывать по производительности в тридцать и более раз у коллективной разработки, что достигается за счет:

- исключения межличностных коммуникаций, связанных с необходимостью порождения и изучения большого количества технологической документации;
- исключения работ по разбиению проекта на составляющие, по распределению их между исполнителями, по координации деятельности исполнителей и контролю за их работой.

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Коллективная разработка

Собрать стадо из баранов легко, трудно собрать стадо из кошек.

С. П. Капица

Одним из основных вопросов коллективной разработки является разделение труда - от равноправных соисполнителей до организации в виде жесткой структуры.

Харлан Миллз [Брукс 1999] предложил организовывать команды (бригады) главного программиста (chief programmer teams), подобные хирургическим бригадам.

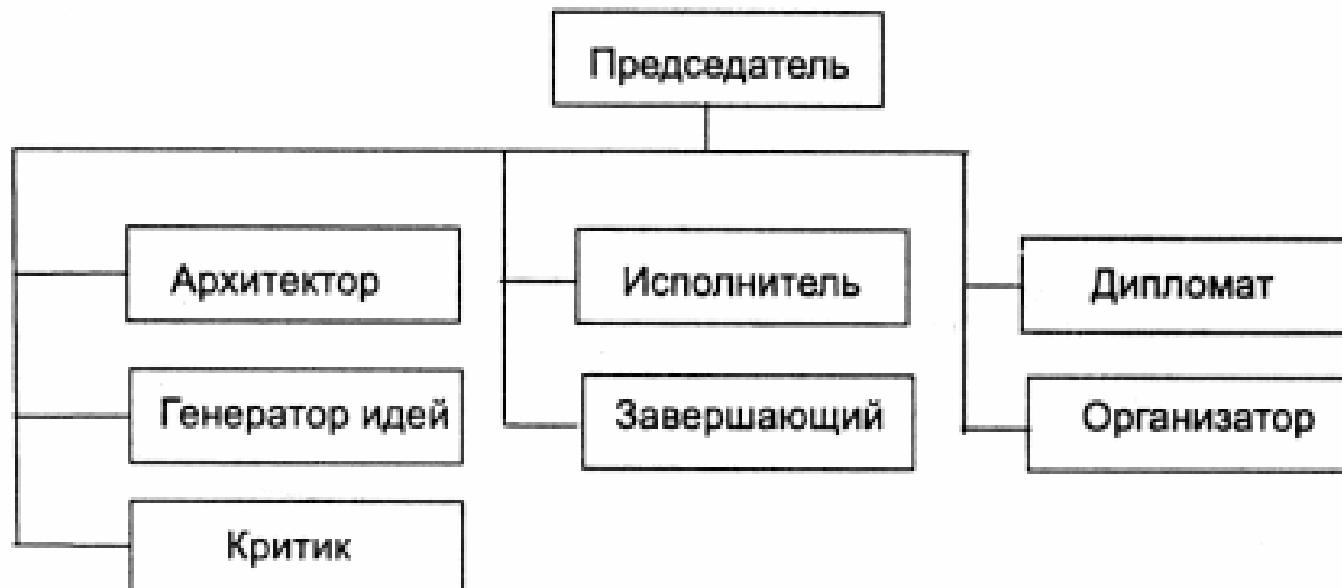


1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Коллективная разработка

Психологические командные роли

Роб Томсет (Rob Thomsett) [Thomsett 1990] предложил восемь ключевых ролей в проекте



1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Коллективная разработка

Психологические командные роли

Роб Томсет (Rob Thomsett) [Thomsett 1990] предложил восемь ключевых ролей в проекте:

Председатель. Выбирает путь, по которому команда движется вперед к общим целям. Умеет обнаружить сильные и слабые стороны команды и обеспечить наибольшее применение потенциала каждого ее участника.

Архитектор. Придает законченную форму действиям команды. Имеет четкое представление о проблемах и их возможных решениях.

Генератор идей. Предлагает радикально новые идеи и стратегии, новые подходы к решению проблем, с которыми сталкивается группа. Особое внимание уделяет главным проблемам.

Критик. Он же скептик, оценивающий проблемы с прагматической точки зрения. Ищет недостатки, изъяны и недоделки. Компенсирует оптимизм генератора идей.

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Коллективная разработка

Исполнитель. Работник, собственно занимающийся написанием кода. Как правило, он не обладает широтой кругозора.

Завершающий. Поддерживает в команде настойчивость в достижении цели. Играет доминирующую роль на завершающих стадиях разработки.

Дипломат. Поддерживает силу духа в участниках проекта. Оказывает им помощь в трудных положениях. Пытается улучшить взаимоотношения в команде.

Организатор. Обнаруживает и сообщает о новых идеях, разработках и ресурсах. Имеет много друзей и связей в своей организации, с помощью которых можно выпросить или одолжить необходимые ресурсы.

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Общинная разработка

Совершенство в проекте достигается не тогда, когда нечего добавить, а тогда, когда нечего убрать.

Антуан де Сент-Экзюпери

Идеология общинной ("базарной") модели разработки сформулирована в программной статье Эрика Раймонда "Собор и Базар".

Характеристики:

- Децентролизованность разработки. Не существует ограничения сверху на количество людей, принимающих участие в проекте.
- Разработка ведется на базе открытых исходных текстов.
- Большое количество внешних тестеров (бета-тестеров), позволяющих быстро обнаруживать ошибки и проблемы в программе.

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Организация проектной команды

Роли и ответственности участников типового проекта разработки ПО можно условно разделить на пять групп:

- Анализ. Извлечение, документирование и сопровождение требований к продукту.
- Управление. Определение и управление производственными процессами.
- Производство. Проектирование и разработка ПО.
- Тестирование ПО.
- Обеспечение. Производство дополнительных продуктов и услуг.

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Сотрудники и роли

- ▶ Виды отношений:
 - Один сотрудник – одна роль
 - Один сотрудник – несколько ролей
 - Несколько сотрудников – одна роль
 - Несколько сотрудников – несколько ролей
- ▶ В общем случае отношение «многие ко многим»
- ▶ Отношение существует только в контексте одного проекта
- ▶ Роли часто могут совмещаться
- ▶ Не все роли присутствуют во всех проектах

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах

▶ Основные

- Заказчик (customer)
- Планировщик ресурсов (planner)
- Менеджер проекта (project manager)
- Архитектор (architect)
- Руководитель команды (team leader, team lead)
- Разработчик (developer)
- Тестер (tester, QA)
- Разработчик документации (technical writer)
- Пользователь (user)
- Инженер группы поддержки (support engineer)

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах

► Дополнительные

- Эксперт предметной области
- Специалист по пользовательскому интерфейсу и эргономике
- Ответственный за выпуск релизов
- Библиотекарь
- ...

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах



Роли в программных проектах

Заказчик

- ▶ Инициирует разработку
- ▶ Участвует в сборе требований
- ▶ Участвует в разработке спецификации требований
- ▶ Принимает результаты разработки



1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах

Планировщик ресурсов

- ▶ Член руководства организации
- ▶ Выдвигает и координирует требования к проектам в организации
- ▶ Развивает и направляет план выполнения проекта с точки зрения организации
- ▶ Обеспечивает финансирование проекта

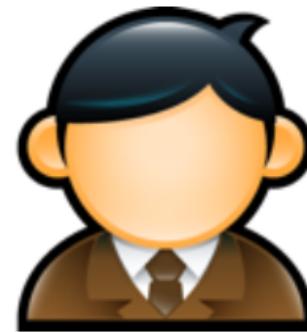


1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах

Менеджер проекта

- ▶ Внешние функции:
 - Взаимодействие с инициатором проекта:
 - Заказчиком
 - Планировщиком ресурсов
- ▶ Внутренние функции:
 - Распределяет задачи среди членов команды
 - Организует выполнение проекта



1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах

Архитектор

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



1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах

Руководитель команды

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



1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах

Разработчик

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



1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах

Тестер

Тестирующий, Quality Assurance (QA)

- ▶ Проверяет качество программного обеспечения (функциональность, надежность, эффективность и т.п.)
Составляет тесты для каждой фазы проектирования продукта
- ▶ Исполняет созданные тесты
- ▶ Выполняет функциональное тестирование
- ▶ Выполняет интеграционное, системное тестирование



1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах

Разработчик документации

Технический писатель, technical writer

- ▶ Разработка программной документации
- ▶ Разработка эксплуатационной документации
- ▶ Ведение информационной поддержки процесса разработки

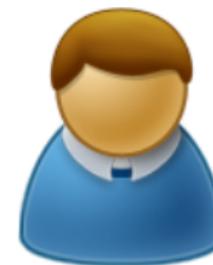


1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах

Пользователь

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



1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах

Эксперт предметной области

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

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах

Специалист по пользовательскому интерфейсу и эргономике

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

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах

Ответственный за выпуск релиза

- ▶ Определяет и реализует политику выпуска релизов
- ▶ Формулирует и проверяет требования к конкретному релизу:
 - Необходимая функциональность
 - Состав релиза
- ▶ Определяет дату выхода релиза
- ▶ Контролирует процесс выхода релиза

1. ГРУППА ПРОЕКТА И РОЛИ УЧАСТНИКОВ

Роли в программных проектах

Совмещение ролей

	Зак	Плн	Мен	Арх	Рук	Разр	Тст	Док	Плз	Под
Зак	-		-	-	-	-	В	Ч	В	-
Плн	-		В	-	-	-	-	-	-	-
Мен	-	В		+	-	-	-	-	-	В
Арх	-	-	+		В	В	-	Ч	-	-
Рук	-	-	-	В		В	-	Ч	-	В
Разр	-	-	-	В	В		X	Ч	-	Ч
Тст	В	-	-	-	-	X		Ч	+	Ч
Док	Ч	-	-	Ч	Ч	Ч	Ч		Ч	+
Плз	В	-	-	-	-	-	+	Ч		-
Под	-	-	В	-	В	Ч	Ч	+	-	

- Ч – часто совмещаются
- + – может совмещаться

- – не может
- X – вредно!

2. ДОКУМЕНТАЦИЯ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Роль документации в проекте:

- средство передачи информации между разработчиками ПС,
 - средство управления разработкой ПС,
 - средство передачи пользователям информации, необходимой для применения и сопровождения ПС.
-
- Документацию процесса разработки можно разбить на две группы:
 - 1. Документы управления разработкой ПС
 - 2. Документы, входящие в состав ПС (product documentation)

2. ДОКУМЕНТАЦИЯ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Документы управления разработкой ПС:

- Планы, Оценки, Расписания
- Отчёты
- Стандарты
- Рабочие документы
- Заметки и переписка

2. ДОКУМЕНТАЦИЯ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Документы, входящие в состав ПС

1.Пользовательская документация ПС.

- Общее функциональное описание ПС
- Руководство по инсталляции ПС
- Инструкция по применению ПС
- Справочник по применению ПС
- Руководство по управлению ПС

2.Документация по сопровождению ПС

- Документация, определяющая строение программ и структур данных ПС и технологию их разработки
- Документация, помогающая вносить изменения в ПС

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

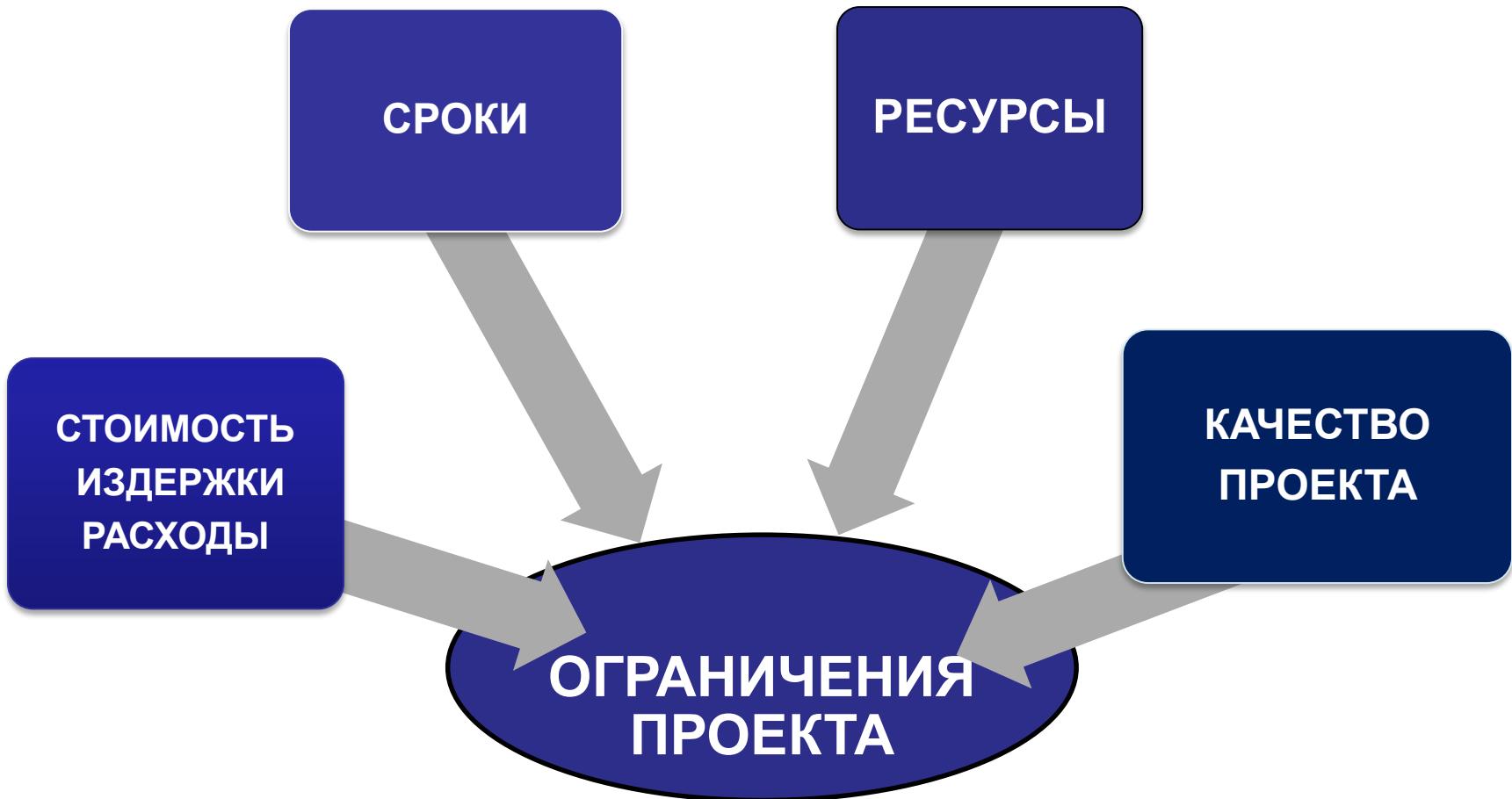
Составляющие управления проектом

- **Проект** – целенаправленное ограниченное во времени мероприятие, направленное на создание уникального продукта или услуги.
- **Управление проектом** заключается в управлении производством программного продукта в рамках отведенных средств и времени.

PMBOK. Руководство к Своду знаний по управлению проектами, 3-е изд., PMI, 2004.

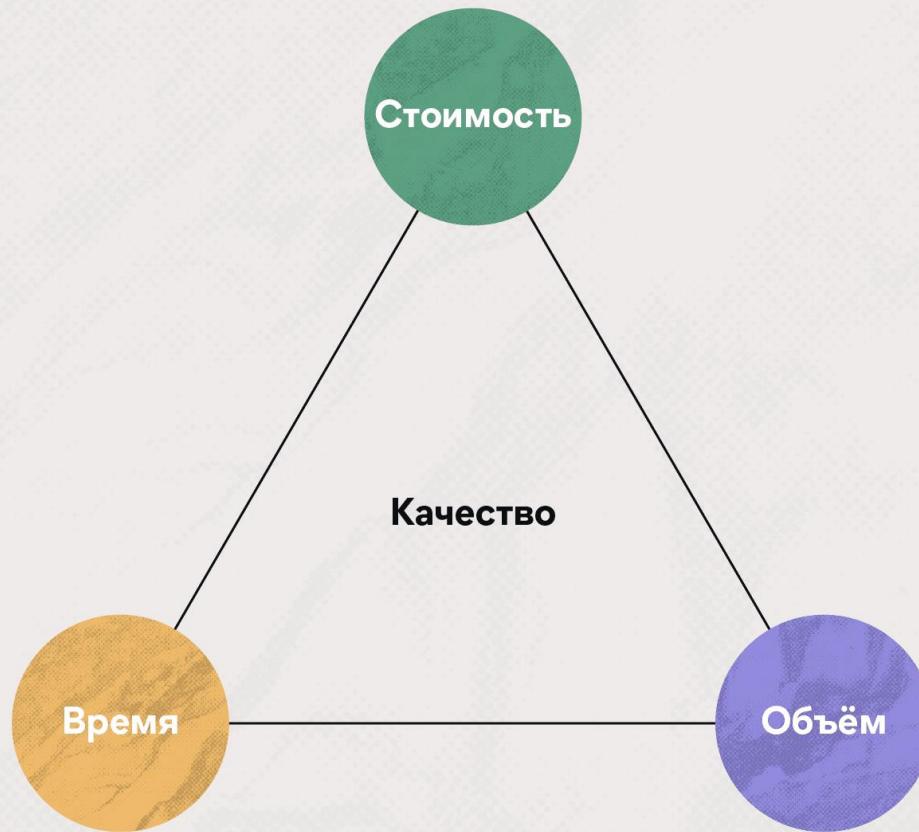
PMI, Project Management Institute, PMBOK — американский национальный стандарт ANSI/PMI 99-001-2004.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ



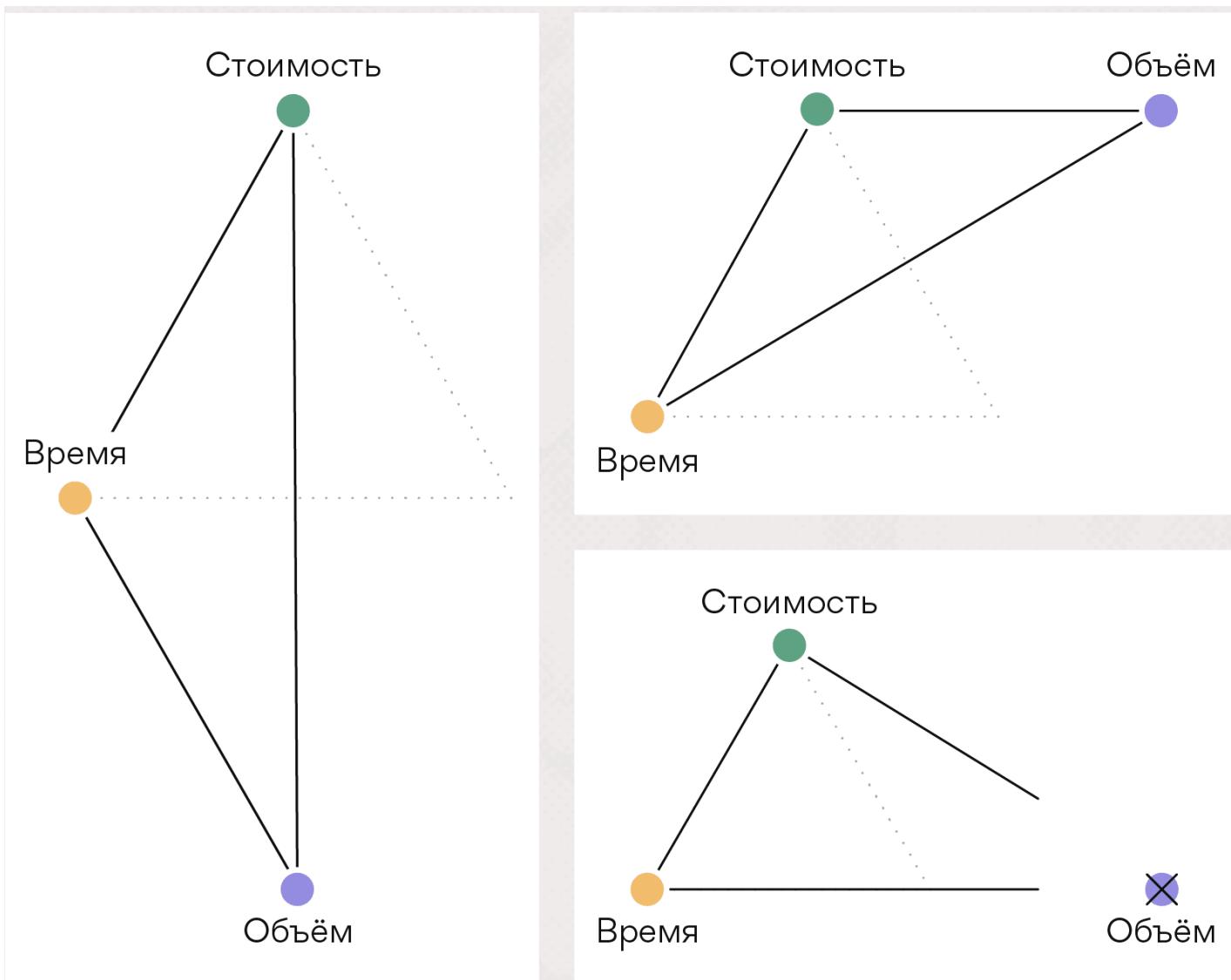
3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Железный треугольник управления проектами



3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Проектный треугольник



3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Составляющие управления проектом

Управление проектом охватывает:

- инфраструктуру (организационные моменты);
- управляющий процесс (права и ответственные участники);
- процесс разработки (методы, инструменты, языки, документация и поддержка);
- расписание (моменты времени, к которым должны быть представлены выполненные фрагменты работы).

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

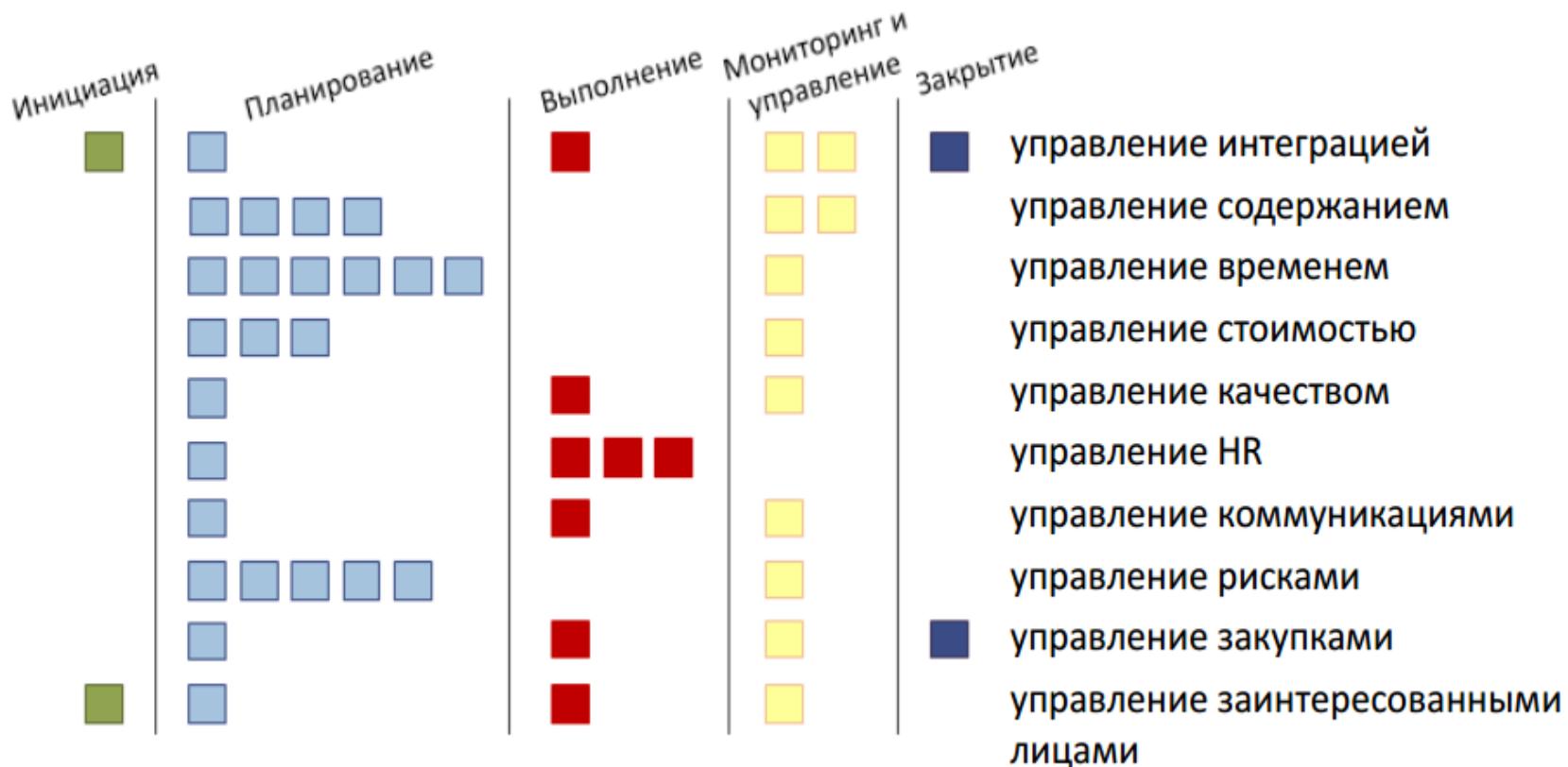
Составляющие управления проектом

Типичная схема процесса управления проектом

1. Понять содержание проекта, область применения и временные рамки.
2. Определиться с процессом разработки.
3. Выделить организационную структуру.
4. Определить управляющий процесс (ответственность участников).
5. Разработать расписание проекта.
6. Разработать план подбора кадров.
7. Начать управление рисками.
8. Определить, какие документы необходимо выработать.
9. Начать процесс разработки.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Структура стандарта



3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Соответствие между процессами и областями знаний

Область знаний	Группа процессов инициации	Группа процессов планирования	Группа процессов исполнения	Группа процессов мониторинга и контроля	Группа процессов закрытия
1. Управление интеграцией проекта	1.1. Разработка устава проекта	1.2. Разработка плана управления проектом	1.3. Руководство и управление работами проекта	1.4. Мониторинг и контроль работ проекта 1.5. Интегрированный контроль изменений	1.6. Закрытие проекта или фазы
2. Управление содержанием проекта		2.1. Планирование управления содержанием 2.2. Сбор требований 2.3. Определение содержания 2.4. Создание ИСР		2.5. Подтверждение содержания 2.6. Контроль содержания	
3. Управление сроками проекта		3.1. Планирование управления расписанием 3.2. Определение операций 3.3. Определение последовательности операций 3.4. Оценка ресурсов операций 3.5. Оценка длительности операций 3.6. Разработка расписания		3.7. Контроль расписания	
4. Управление стоимостью проекта		4.1. Планирование стоимостью проекта 4.2. Оценка стоимости 4.3. Определение бюджета		4.4. Контроль стоимости	
5. Управление закупками проекта		5.1. Планирование управления закупками	5.2. Проведение закупок	5.3. Контроль закупок	5.4. Закрытие закупок

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Области знаний

Область знаний	Группа процессов инициации	Группа процессов планирования	Группа процессов исполнения	Группа процессов мониторинга и контроля	Группа процессов закрытия
6. Управление коммуникациями проекта		6.1. Планирование управления коммуникациями проекта	6.2. Управление коммуникациями	6.3. Контроль коммуникаций	
7. Управление заинтересованными сторонами проекта	7.1. Определение заинтересованных сторон	7.2. Планирование управления заинтересованными сторонами	7.3. Управление вовлечением заинтересованных сторон	7.4. Контроль вовлечение заинтересованных сторон	
8. Управление человеческими ресурсами проекта		8.1. Планирование управления человеческими ресурсами	8.2. Набор команды проекта 8.3. Развитие команды проекта 8.4. Управление команды проекта		
9. Управление качеством проекта		9.1. Планирование управления качеством	9.2. Обеспечение качества	9.3. Контроль качества	
10. Управление рисками проекта		10.1. Планирование управления рисками 10.2. Идентификация рисков 10.3. Качественный анализ рисков 10.4. Количественный анализ рисков 10.5. Планирование реагирования на риски		10.6. Контроль рисков	

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Процессы управления проектами

- процессы инициации — принятие решения о начале выполнения проекта
- процессы планирования — определение целей и критериев успеха проекта, разработка плана
- процессы реализации — координация людей и других ресурсов для выполнения плана
- процессы контроля — определение соответствия плана и исполнения проекта поставленным целям и критериям успеха, принятие решений о необходимости применения корректирующих воздействий
- процессы завершения — формализация выполнения проекта и подведение его к упорядоченному финалу

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Схема жизненного цикла IT-проекта

Жизненный цикл – это последовательность фаз проекта, через которые он должен пройти для гарантированного достижения целей проекта



3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Разработка устава проекта

1. Краткое описание проекта

1.1 Название проекта

<Название проекта>

1.2 Суть проекта

<Что представляет собой проект? Это разработка или внедрение? Разработка чего и для чего – одним/двумя предложениями>

1.3 Бизнес-окружение проекта

<Почему предпринят проект; каковы ожидания / предположения высшего менеджмента? Как цели проекта связаны со стратегическими целями клиента (продукт выведет его на новые рынки, позволит сэкономить и т.п.)?

1.4. Цели проекта

<Цели проекта SMART (перечисляемым списком)>

1.5. Риски проекта

<Если известно - предполагаемые результаты предполагаемой качественной оценки совокупных рисков проекта (негативных и позитивных) – высокий / средний / низкий. Краткое обоснование оценки (1-2 предложения)>

<Если известны – главные высокоуровневые риски (перечислить)>

Устав Проекта

2. Описание продукта и поставок

2.1 Продуктом проекта является (перечень поставок)

<Перечислить результаты поставки списком - дистрибутив ПО, пользовательская документация; и т.п.>

2.2 Главными требованиями к продукту являются (продукт позволяет):

<Указать высокоуровневые требования (функциональные и нефункциональные)>

2.3 Требованиями к продукту НЕ являются (продукт не включает):

<Указать значимые высокоуровневые требования, не включаемые в настоящий проект>

2.4 Правила приемки поставок:

<Указать общие правила приемки (будет ли создаваться комиссия, на основании каких документов планируется сдача)>

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Разработка устава проекта

3. Ограничения проекта

3.1 Вехи и дата завершения проекта:

Начало проекта	<Дата>
• <Указать название вехи 1>	<Дата>
• <Указать название вехи 1>	<Дата>
• <...>	<...>
Завершение проекта	<Дата>

3.2 Общий бюджет проекта:

<Бюджет включает все расходы по проекту + все расходы по управлению рисками, в том числе и управленческие резервы>

3.3 Ограничения по выполнению и организации работ

<Например, для успеха проекта критично важно, чтобы сотрудники исполнителя не общались с определенным департаментом заказчика; или указом менеджмента должна быть выбрана конкретная аппаратная платформа>

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Разработка устава проекта

4. Руководитель проекта и его полномочия

4.1 Назначенный руководитель проекта

<ФИО руководителя проекта - представителя исполнителя, ответственного за реализацию проекта в срок, в пределах бюджета и с заданным качеством>

4.2 Полномочия руководителя проекта

<Как руководитель проектов будет формировать команду – может ли он брать любых людей, должен ли он это обосновать финансово? Будет ли у руководителя проекта помощник?>

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

5. Заинтересованные лица и ресурсы

5.1 Заказчик проекта

<ФИО, должность, и название организации>

5.2 Ключевые пользователи результатов проекта:

<Перечень лиц или организаций>

5.3 Спонсор проекта

<ФИО, должность, и название организации>

5.4 Куратор проекта

<Если определен куратор – указать. Куратор это промежуточное звено между ПМ и спонсором, уполномоченный принимать решения о выделении ресурсов и изменениях в проекте. Куратор может быть сотрудником проектного офиса (PMO), если таковой развернут в организации>

5.5 Команда проекта

<Сколько людей выделено на проект? Из каких департаментов? Какие люди / под какие работы – будут наняты извне (в штат / по субконтракту), если это известно? Какие вводные по привлекаемым субподрядчикам известны сейчас? Какие вводные по требуемой квалификации известны? Ограничен ли бюджет на привлечение сотрудников?>

5.6 Инфраструктура

<Какие требования к специальному оборудованию, используемому на проекте известны? Потребуется ли лицензионное программное обеспечение для производства продукта? Ограничен ли бюджет на инфраструктурные ресурсы?>

5.7 Соисполнители проекта

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Разработка устава проекта

6. Согласовательные подписи			
УТВЕРЖДАЮ:			
Имя	Должность	Подпись	Дата (MM/DD/YYYY)
<Устав обязательно подписывается спонсором>			

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Пример

1.1 Название проекта

1.2 Цель проекта - повышение эффективности основной производственной деятельности отдела «123».

1. 3. Дополнительными целями проекта являются:

1. 3.1.Установление долгосрочных отношений с важным заказчиком ОАО «XYZ».

1. 3. 2.Выход на новый перспективный рынок современных B2C систем.

1.4. Повышение эффективности основной производственной деятельности отдела «123» достигается :

- снижением затрат на обработку заявок
- снижением сроков обработки заявок
- повышением оперативности доступа к информации о наличии продукции

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Пример

- повышением оперативности доступа к информации о прохождении заявок
 - повышением надежности и полноты хранения информации о поступивших заявках и результатах их обработки.
- результатах их обработки.

2. Продуктами проекта являются:

- 2.1. Прикладное ПО и документация пользователей.
- 2.2. Базовое ПО.
- 2.3. Оборудование ЛВС, рабочие станции, сервера и операционно-системное ПО.
- 2.4. Проведение пуско-наладочных работ и ввод в опытную эксплуатацию.
- 2.5. Обучение пользователей и администраторов системы.
- 2.6. Сопровождение системы на этапе опытной эксплуатации.
- 2.7. Передача системы в промышленную эксплуатацию.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Пример

3. Система должна автоматизировать следующие функции:
 - 3.1. Авторизация и аутентификация пользователей.
 - 3.2. Просмотр каталога продуктов.
 - 3.3. Поиск продуктов по каталогу.
 - 3.4. Заказ выбранных продуктов.
 - 3.5. Просмотр информации о статусе заказа.
 - 3.6. Информирование клиента об изменении статуса заказа.
 - 3.7. Просмотр и обработка заказов исполнителями из службы продаж.
 - 3.8. Просмотр статистики поступления и обработки заказов за период.
 - 3.9. Подготовка и сопровождение каталога продукции.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Пример

4. В рамки проекта не входят:

4.1. Защита системы от преднамеренного взлома.

4.2.Разработка B2B API и интеграция с другими системами.

5. Риски проекта

Задачи системы поняты недостаточно полно. Понимание масштаба и рамок проекта недостаточно. Системы создаются на новой технологической платформе, сомнения в рыночной стабильности платформы. Суммарный уровень рисков следует оценить выше среднего..

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Пример

6.1. Сроки проекта

6.1.1. **03.03** старт

6.1.2 **28.11** завершение

6.1. 3 Контрольные точки:

15.04 ТЗ утверждено

30.04 1-я итерация завершена. Подсистема заказа документации передана в тестовую эксплуатацию (на серверах разработчика).

5.05 Монтаж оборудования у заказчика завершен....

.....

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Пример

6.2. Расходная часть бюджета проекта

6.2.1. Разработка и сопровождение прикладного ПО:

6.2.1.1. $9000 \text{ чел.} * \text{час.} * \$40 = \$360\,000$

6.2.2. Поставка оборудования и операционно-системного ПО:

6.2.2.1. Сервера $* \$10\,000 = \$30\,000$

6.2.2.2. Поставка базового ПО:

Oracle RDBMS \$20 000

.....

Итого:

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Ключевые участники и заинтересованные стороны

- Спонсор проекта - лицо или группа лиц, предоставляющая финансовые ресурсы для проекта в любом виде.
- Заказчик проекта - лицо или организация, которые будут использовать продукт, услугу или результат проекта.
- Пользователи результатов проекта.
- Куратор проекта - представитель исполнителя, уполномоченный принимать решение о выделении ресурсов и изменениях в проекте
- Руководитель проекта - представитель исполнителя, ответственный за реализацию проекта в срок, в пределах бюджета и с заданным качеством.
- Соисполнители проекта. Субподрядчики и поставщики

Процессы планирования

Описание продукта

Устав проекта

Создание WBS

Содержание проекта

Иерархическая структура работ (WBS)

Создание расписания

Сетевая диаграмма

Пул ресурсов

Расписание проекта

Создание Плана проекта

Бюджет проекта

Планы управления сроками,
стоимостью ...

WBS (*Work Breakdown Structure*) - Иерархическая структура
работ

Определение содержания

Декомпозиция – последовательное деление основных результатов проекта на меньшие компоненты до уровня детализации, достаточного для обеспечения управления проектом (планирования, исполнения, контроля и завершения)

Корректность декомпозиции

- Каждый элемент **полностью определен**
- Нижнеуровневые операции и **необходимы, и достаточны для соответствующего элемента**
- Каждый элемент нижнего уровня **может быть оценен по** стоимости, срокам и приписан к организационной единице (подразделение, исполнитель), которая способна его выполнить

Определение содержания Декомпозиция - метод



- Каждая цель имеет свой код в иерархии дерева целей.
- Каждая работа имеет свой код в иерархии процесса.
- Лучше планировать цели так, чтобы подзадач было не более 7 (5+2).

WBS ур. 1

1 Велосипед 100

WBS ур. 2

- 1.1 Набор для рамы 15
- 1.2 Ведущие звёзды 5
- 1.3 Колёса 30
- 1.4 Тормозная система 5
- 1.5 Переключ. скоростей 5
- 1.6 Сборка 35
- 1.7 Управл-е проектом 5

WBS ур. 3

1.1 Набор для рамы

- 1.1.1 Рама 7
- 1.1.2 Руль 2
- 1.1.3 Вилка 3
- 1.1.4 Сиденье 3

1.2 Ведущие звёзды 5

1.3 Колёса

- 1.3.1 Пер. колесо 13
- 1.3.2 Зад. колесо 17

1.4 Тормозная система 5

1.5 Переключ. скоростей 5

1.6 Сборка

- 1.6.1 Концепт 3
- 1.6.2 Дизайн 5
- 1.6.3 Монтаж 10
- 1.6.4 Тестирование 17

1.7 Управл-е проектом 5

Планирование работ

- *Иерархическая структура работ (work breakdown structure)* – иерархическое разбиение всей работы, которую необходимо выполнить для достижения целей проекта, на более мелкие операции, до такого уровня, на котором способы выполнения этих действий вполне ясны и соответствующие работы могут быть оценены и спланированы.
- Операция – работа, задача

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Планирование работ

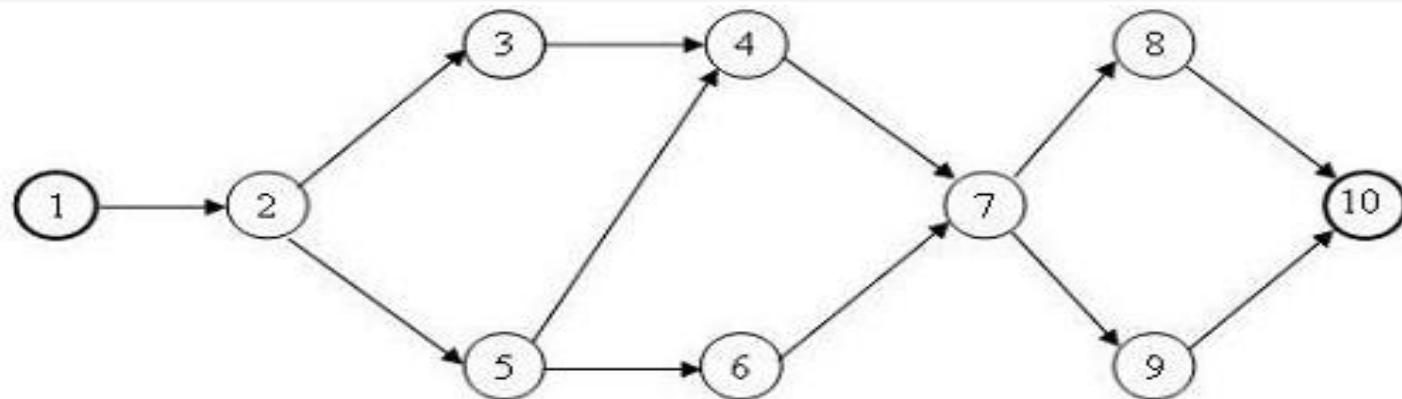
- ***Разработка графика проекта***

- Уточнить имеющуюся структуру работ проекта
- Установить зависимости между отдельными работами
- Оценить время выполнения и трудоемкость каждой из работ
- Построить *сетевой график* проекта

Сетевой график – это ориентированный граф, в котором вершинами обозначены работы проекта, а дугами – временные взаимосвязи работ.

- Выделить *критический путь* — последовательностей работ от начала до конца проекта, сумма длительностей которых максимальна среди таких последовательностей.

Номер работы	Название работы	Длительность
1	Начало реализации проекта	0
2	Постановка задачи	10
3	Разработка интерфейса	5
4	Разработка модулей обработки данных	7
5	Разработка структуры базы данных	6
6	Заполнение базы данных	8
7	Отладка программного комплекса	5
8	Тестирование и исправление ошибок	10
9	Составление программной документации	5
10	Завершение проекта	0



3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Планирование работ

- ***Разработка графика проекта***
 - Построить календарный график проекта (диаграмму Ганта), который отражает:
 - структуру работ, полученную на основе сетевого графика;
 - состав используемых ресурсов и их распределение между работами;
 - календарные даты, к которым привязываются моменты начала и завершения работ.

Последовательность операций

- **Последовательность операций** - идентификация и документирование логических взаимосвязей между операциями

Входы

- Перечень операций
- Зависимости
- **Контрольные события (milestones)** – операции с в результате исполнения которых достигаются промежуточные цели

Методы

- Диаграммные методы

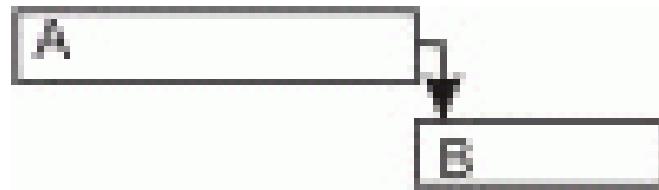
Результаты

- Сетевая диаграмма проекта

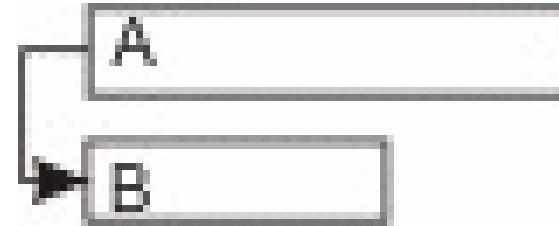
3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Типы зависимостей

- Finish-to-start (Окончание-начало), или сокращенно FS (ОН), задача В не может начаться, пока не завершена задача А.



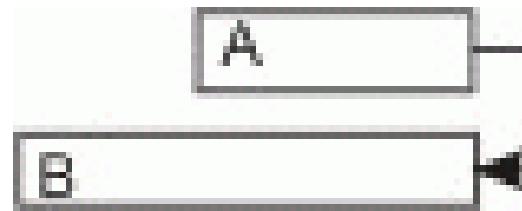
- Start-to-start (Начало-начало), или сокращенно SS (НН), задача В не может начаться до тех пор, пока не началась задача А.



3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Типы зависимостей

- Finish-to-Finish (Окончание-окончание), или сокращенно FF (00), задача В не может закончиться до тех пор, пока не закончилась задача А.



- Start-to-Finish (Начало-окончание), или сокращенно SF (HO), задача В не может закончиться до тех пор, пока не началась задача А.



Ресурсы в проекте

Ресурс – это трудовая, материальная, финансовая, техническая или иная единица, которая используется для выполнения задач проекта.

Виды ресурсов:

- *Трудовые* – возобновляемые ресурсы (исполнители, машины и оборудование)
- *Материальные* – не возобновляемые ресурсы (материалы и энергоносители)
- *Затратные* – различные виды денежных расходов сопряженных с работами проекта, которые напрямую не зависят от объема и длительности работ.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Характеристики ресурсов

- Доступность
- Стоимость:
 - Стандартная ставка
 - Стоимость использования
- *Универсальный ресурс* (прототип ресурса) используется для определения требований к персоналу для проекта, например к тестировщикам и разработчикам.
- *Выделенный ресурс* – ресурс, формально выделенный для любого назначения задач, имеющегося в проекте (по умолчанию). Выбор данного типа резервирования влияет на доступность и загрузку ресурса.
- *Предложенный ресурс* - ресурс, ожидающий выделения ресурсов для еще не утвержденного назначения задачи. Такое назначение ресурса не уменьшает его доступности для работы по другим проектам.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Характеристики операций

- ▶ *Длительность задачи (Duration)* — это период рабочего времени, который необходим для выполнения задачи.
- ▶ *Трудозатраты (Work)* — время, затраченное сотрудниками на получение результата.

Назначение

- **Назначение** – это сопоставление задаче перечня трудовых, материальных или затратных ресурсов, которые будут задействованы при ее выполнении.
- **Объем назначения** – доля (процент) рабочего времени по индивидуальному календарю ресурса.

Трудозатраты задачи

$$T = L \cdot \sum V \cdot H$$

где L – длительность задачи, V – объем назначений ресурса, H – ежедневная длительность работы ресурса в часах, сумма берется по всем назначенным задаче трудовым ресурсам.

Типы задач

- ***Fixed Duration*** (Фиксированная длительность, ФД)
- ***Fixed Units*** (Фиксированный объем ресурсов, ФОР)
- ***Fixed Work*** (фиксированные трудозатраты, ФТ)

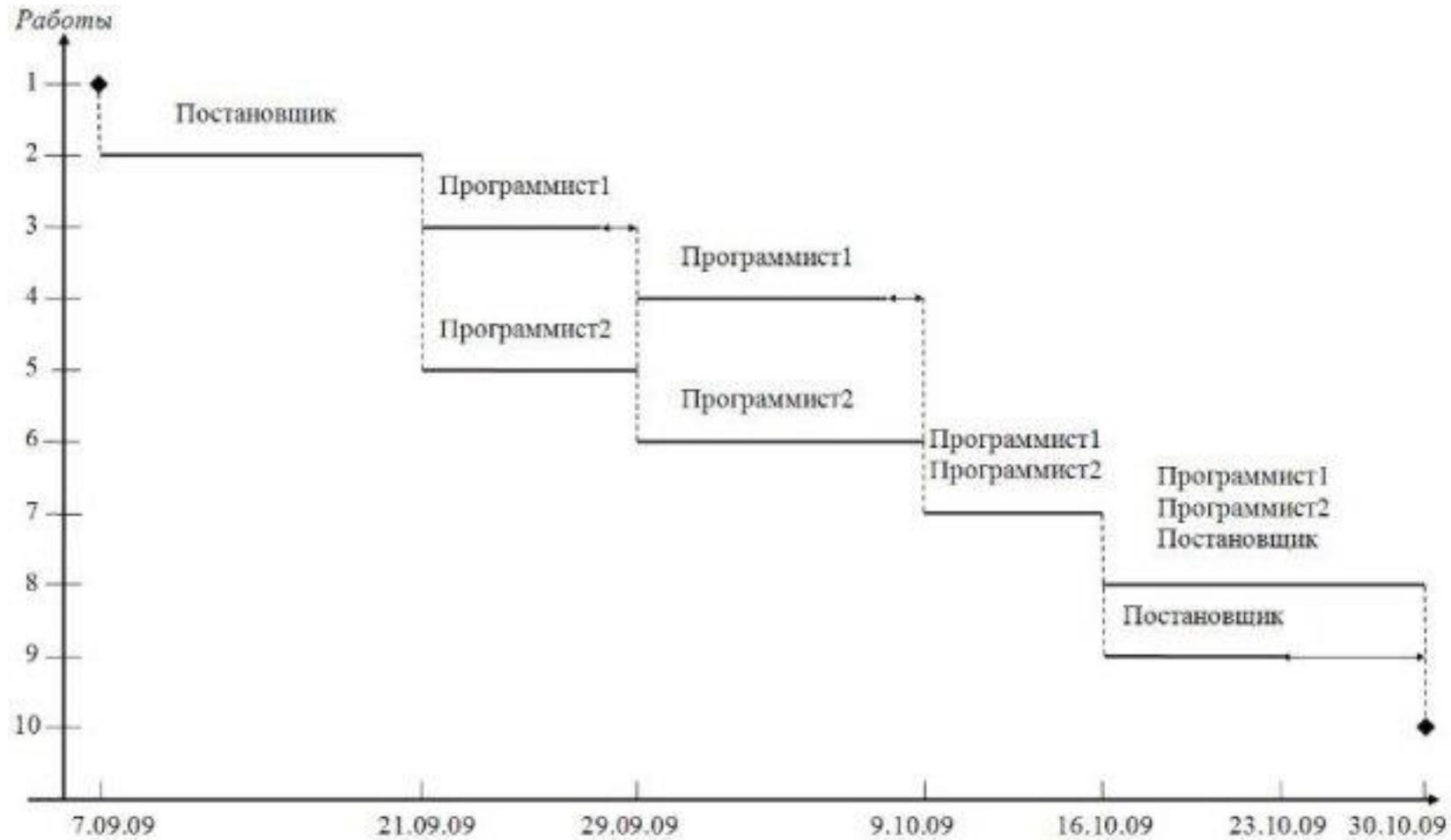
Взаимосвязь свойств для задач разных типов

	Длительности	Трудозатрат	Объема назначения ресурса	Состава ресурсов
Изменяется				
ФОР	Трудозатраты	Длительность	Длительность	Трудозатраты
ФД	Трудозатраты	Объем назначения	Трудозатраты	Трудозатраты
ФТ	Объем назначения	Длительность	Длительность	Длительность

Фиксированный объем работ

	Длительности	Трудозатрат	Объема назначения ресурса	Состава ресурсов
Изменяется				
ФОР	Трудозатраты	Длительность	Длительность	Длительность
ФД	Трудозатраты	Объем назначения	Трудозатраты	Трудозатраты

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ



Календарный график проекта

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Планирование работ

№	Название задачи	Длительность	Начало	Окончание	Пред.	28 Июл '08	04 Авг '08	11 Авг '08
						П	В	С
1	Начало реализации проекта	0 дней	Ср 30.07.08	Ср 30.07.08				
2	■ Программирование	3 дней?	Ср 30.07.08	Пт 01.08.08				
3	Постановка задачи	1 день?	Ср 30.07.08	Ср 30.07.08	1			
4	Разработка интерфейса	1 день?	Чт 31.07.08	Чт 31.07.08	3			
5	Разработка модулей обработки данных	1 день?	Пт 01.08.08	Пт 01.08.08	4;6			
6	Разработка структуры базы данных	1 день?	Чт 31.07.08	Чт 31.07.08	3			
7	Заполнение базы данных	1 день?	Пт 01.08.08	Пт 01.08.08	6			
8	Программирование завершено	0 дней	Пт 01.08.08	Пт 01.08.08	5;7			
9	■ Отладка	5 дней?	Пн 04.08.08	Пт 08.08.08				
10	Отладка программного комплекса	1 день?	Пн 04.08.08	Пн 04.08.08	8			
11	Тестирование и исправление ошибок	1 день?	Пт 08.08.08	Пт 08.08.08	100%			
12	Составление программной документации	1 день?	Пт 08.08.08	Пт 08.08.08	10			
13	Отладка завершена	0 дней	Пт 08.08.08	Пт 08.08.08	11;1			
14	Конец проекта	0 дней	Пт 08.08.08	Пт 08.08.08	13			

The Gantt chart illustrates the project timeline. Phase 1 (July 30 - August 1) includes tasks 1 through 8. Phase 2 (August 4 - 8) includes tasks 9 through 13. Task 14 is at the end. Milestones are marked with diamonds: 30.07, 01.08, 04.08, 08.08, and 08.08.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

- **Риски** проекта – неопределенности, негативно влияющие на проект, вплоть до полного провала.

Риски проекта, влияющие на его ход:

- **Технологические риски** — недостаточная производительность и гибкость используемых технологий и инструментов.
- **Кадровые риски** — вероятность не набрать команду или набрать неподходящую, возможность отсутствия у ее членов необходимых навыков, возможность их низкой производительности, вероятность возникновения серьезных конфликтов.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

Риски проекта, влияющие на его ход:

- **Риски требований** — возможность изменений в требованиях к результатам.
- **Коммерческие риски** — вероятность неправильной оценки коммерческих составляющих проекта: неверной оценки рынков сбыта, времени и стоимости проекта; возможность непредвиденных расходов.
- **Управленческие риски** — вероятность выбора неправильных целей проекта, недостаточного контроля состояния проекта, возможность принятия неправильных решений и неэффективных мер.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

Риски проекта, влияющие на его ход:

- **Производственные риски** — невозможность или нерентабельность производства продукции и сбои в производстве. При производстве ПО достаточно малы, сводятся к сбоям в изготовлении коробок с продуктом.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

Риски продукта, влияющие на результаты проекта :

- **Технические риски** — возможность неуспеха в достижении запланированных показателей качества результатов проекта, вероятность вообще не получить нужный результат.
- **Эксплуатационные риски** — невозможность использования продукта или неготовность организаций- пользователей к его эксплуатации.
- **Правовые и общественные риски** — возможность возникновения патентных споров, конфликтов с коммерческими, общественными и государственными организациями по поводу самого продукта или его использования.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

Бизнес-риски :

- **Контрактные риски** — ненадежность соисполнителей, (суб)подрядчиков и поставщиков, возможность возникновения юридических претензий.
- **Инвестиционные риски** — вероятность отказа или задержек в финансировании со стороны части или всех инвесторов проекта.
- **Сбытовые риски** — возможность неполучения запланированных доходов от реализации результатов проекта, отказа пользователей от продуктов, сбоев в каналах сбыта.
- **Конъюнктурные риски** — возможность опережения проекта аналогичными проектами конкурентов, блокады ими рынка, непредвиденной конкуренции.

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

- Идентификация
- Разработка стратегии преодоления рисков
- Выбор приоритетов
- Устранение или уменьшение риска

Метод расчета приоритета рисков

Вероятность (1-10) 1- маловероятно	Ущерб (1-10) 1 – слабое влияние	Стоимость устранения 1 - низкая	Расчет приоритета
Высокопри- оритетный	10	10	1 $(11-10) \times (11-10) \times 1 = 1$
Низкопри- оритетный	1	1	10 $(11-1) \times (11-1) \times 1 = 1000$

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

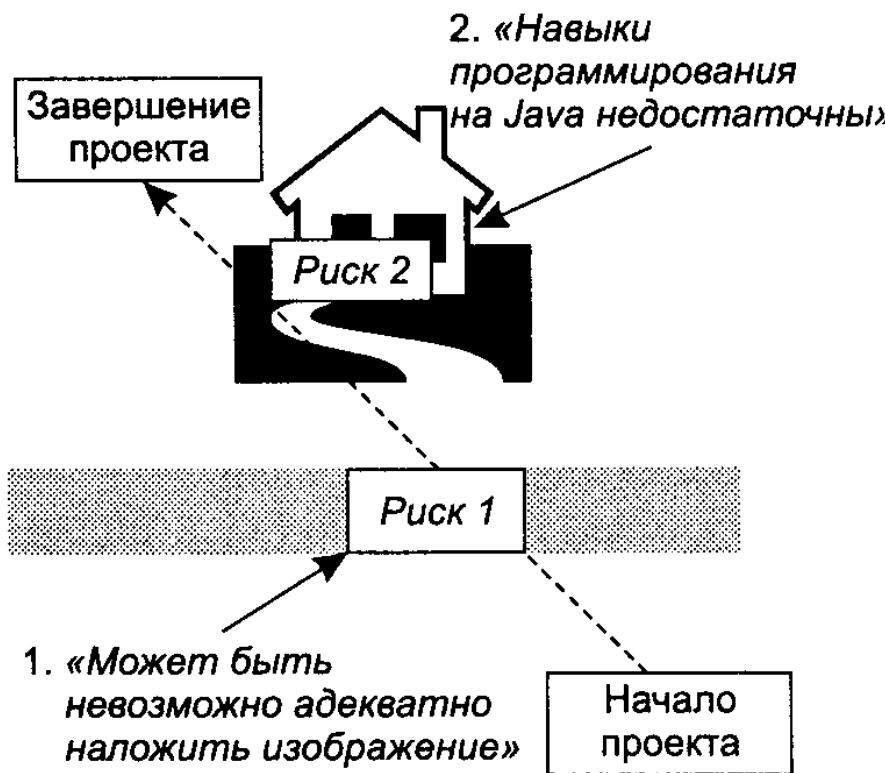
Вида стратегий преодоления рисков:

- *Стратегии предотвращения или обхода рисков.* Они направлены на снижение вероятности риска или полное избавление от него.
- *Стратегии минимизации ущерба.*
- *Планирование реакции на возникающие проблемы.*

3. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Управление рисками

Идентификация



Предупреждение



БГУИР

Факультет компьютерных систем и сетей

**Кафедра программного обеспечения
информационных технологий**

**ОБЩИЕ СВЕДЕНИЯ
О ТРЕБОВАНИЯХ.**

СИНТЕЗ И АНАЛИЗ ТРЕБОВАНИЙ

План лекции:

1. Понятие и классификация требований.
2. Моделирование бизнес-процессов.
Методология функционального
моделирования SADT

Требования: определение

Требования (requirements) – это подробное изложение функционального наполнения системы. Требования должны быть независимы от внутренней архитектуры приложения, т.е. должны описывать то, что необходимо заказчику без деталей реализации (принцип «**what, not how**»).

Требования к продукту и процессу

Требования к продукту. Цель заказчика - получить хороший конечный продукт: функциональный и удобный в использовании.

Требования к процессу. Вопросы формулирования требований к процессу, т.е. к тому, как разработчик будет выполнять работы по созданию целевой системы.

Основные типы требований

Функциональные требования определяют, «что система должна делать».

Нефункциональные требования определяют, «с соблюдением каких условий» должны происходить действия системы (например, скорость отклика при выполнении заданной операции).

Уровни требований

Уровень	Требования В чём (как) выражаются	Что описывают
1	<u>Бизнес требования</u> Общее видение и обзорная документация	Зачем вообще нужен ПП и что с его помощью будет делаться. Например "Нам нужен инструмент, извлекающий бизнес-информацию из различных источников и представляющий её в виде диаграмм и таблиц"
2	<u>Пользовательские требования</u> Use Cases	Задачи, которые пользователь может выполнять с помощью ПП, и реакцию ПП. Например: "Когда пользователь заходит в систему, должен появляться текст приветствия"
3	<u>Функциональные и нефункциональные требования</u> Требования к ПО	Функциональные требования описывают поведение системы, например: "В процессе инсталляции программа должна проверять остаток свободного места на всех несъёмных накопителях". Нефункциональные требования описывают специфические свойства системы, такие как: требования к удобству использования, безопасности, надёжности, расширяемости и т.д. Например: "При одновременной работе с системой 1000 пользователей, минимальное время между возникновением сбоя должно быть более или равно десяти тысячам часов".

Группа функциональных требований

Бизнес-требования (business requirements) – определяют высокоуровневые цели организации или клиента (потребителя) – заказчика разрабатываемого программного обеспечения.

Пользовательские требования (user requirements) – описывают цели/задачи пользователей системы, которые должны достигаться/выполняться пользователями при помощи создаваемой программной системы.

Группа функциональных требований

Функциональные требования (functional requirements) – определяют функциональность (поведение) программной системы, которая должна быть создана разработчиками для предоставления возможности выполнения пользователями своих обязанностей в рамках бизнес-требований и в контексте пользовательских требований.

Нефункциональные требования

Бизнес-правила (business rules) – включают или связаны с корпоративными регламентами, политиками, стандартами, законодательными актами, внутрикорпоративными инициативами, учётными практиками, алгоритмами вычислений и т.д.

Внешние интерфейсы (external interfaces) – пользовательские интерфейсы.

Нефункциональные требования

Атрибуты качества (quality attributes) – описывают дополнительные характеристики продукта в различных измерениях, важных для пользователей и/или разработчиков.

Ограничения (constraints) – формулировки условий, модифицирующих требования или наборы требований, сужая выбор возможных решений по их реализации.



Характеристики требований

Каждое требование должно быть:

Завершённым (complete). Все важные аспекты должны быть включены. Ничто не должно быть оставлено «для будущего определения» (2BD – to be defined).

Непротиворечивым (consistent). Требование не должно содержать противоречий как внутри себя, так и с другими требованиями.

Корректным (correct). Требование должно чётко указывать на то, что должно выполнять приложение.

Недвусмысленным (unambiguous). Требование не должно допускать разнотений.

Проверяемым (verifiable). Требование должно быть сформулировано так, чтобы существовали способы однозначной проверки – выполнено требование или нет.

Характеристики набора требований

Наборы требований должны быть:

Модифицируемыми (modifiable). Структура и стиль набора требований должны быть такими, чтобы набор требований можно было легко модифицировать. Должна отсутствовать избыточность. Должно быть построено корректное содержание всего документа.

Прослеживаемыми (traceable). У каждого требования должен быть уникальный идентификатор, по которому на это требование можно сослаться.

Характеристики набора требований

Наборы требований должны быть:

Проранжированными по важности, стабильности и срочности (*ranked for importance, stability and priority*). Для каждого требования должен быть указан уровень его важности (насколько оно важно для заказчика), стабильности (насколько высока вероятность, что это требование ещё будет изменено в процессе обсуждения деталей проекта) и срочности (как быстро требование должно быть реализовано).

Проблемы с требованиями

Проблемы незавершённости (неполноты). Упущены нефункциональные составляющие требования. Присутствуют скрытые предположения. Присутствуют слишком общие утверждения. **Примеры требований?**

Проблемы противоречивости. Требования противоречат явным образом друг другу (**пример?**) или части требований противоречат друг другу. Требование противоречит здравому смыслу

Проблемы некорректности. К некорректности приводят опечатки, последствия использования сору-paste. Некорректными являются требования, приводящие к избыточным расходам на разработку, при этом не дающие соответствующей отдачи. Технически невыполнимые требования.

Проблемы с требованиями

Проблемы двусмысленности. Требование должно однозначно трактоваться.

Проблемы непроверяемости. «В приложении должно быть ноль ошибок» – это гарантировать невозможно.

Проблемы непроранжированности. «Все требования одинаково важны».

хорошие требования (и наборы требований) всегда проранжированы по трём показателям:

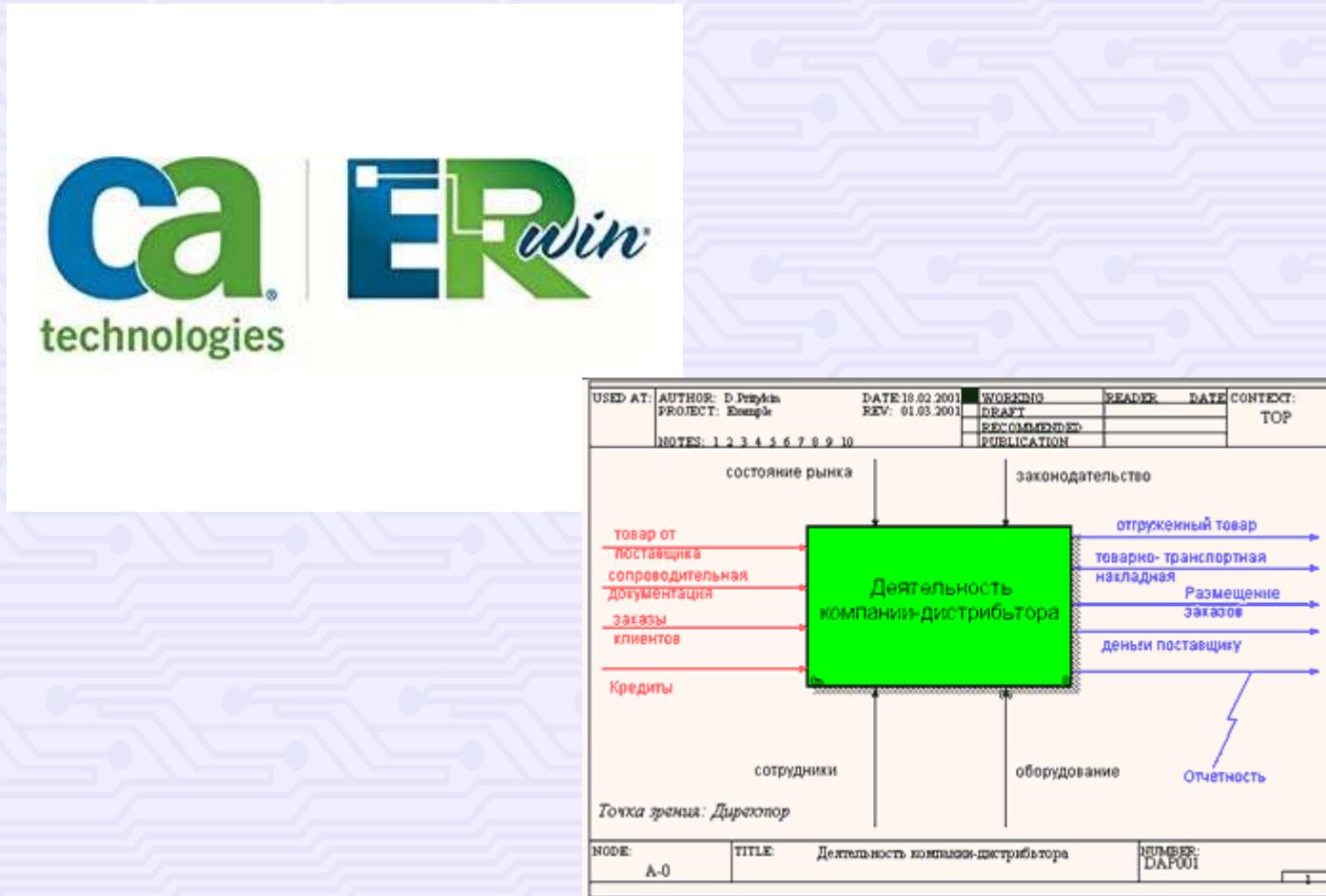
- **важности** (насколько требование важно);
- **приоритетности** выполнения (как быстро его надо реализовать);
- **стабильности** (какова вероятность, что требование изменится).

Способы выявления требований

Основными способами выявления требований являются:

- Интервью
- Наблюдение
- Самостоятельное описание
- Семинары
- Прототипирование

2 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ. МЕТОДОЛОГИЯ SADT



Методология SADT

Методология SADT (Structured Analysis and Design Technique) была создана и опробована на практике в период с 1969 по 1973 гг.

Методология SADT поддерживается Министерством обороны США, которое было инициатором разработки стандарта IDEF0 (Icam DEFinition) - подмножества SADT, являющегося основной частью программы ICAM (Integrated Computer Aided Manufacturing - интегрированная компьютеризация производства), проводимой по инициативе BBC США.

Методология SADT

- Первым элементом SADT модели является ее цель.
- Вторым элементом SADT модели является "точка зрения" - позиция, с которой наблюдается система и создается ее модель.
- Третьим элементом SADT модели является система, для которой составляется модель.

Определение системы формулируется на уровне «черного ящика» т.е. устанавливаются границы системы и определяются взаимосвязи системы с окружающей средой.

Функциональная модель

отображает функциональную структуру системы, т.е. производимые действия и связи между этими действиями.

Концепции:

- графическое представление блочного моделирования.
Функция отображается в виде блока (SA-блок), а интерфейсы входа-выхода дугами;
- строгость и точность;
- отделение организации от функции.

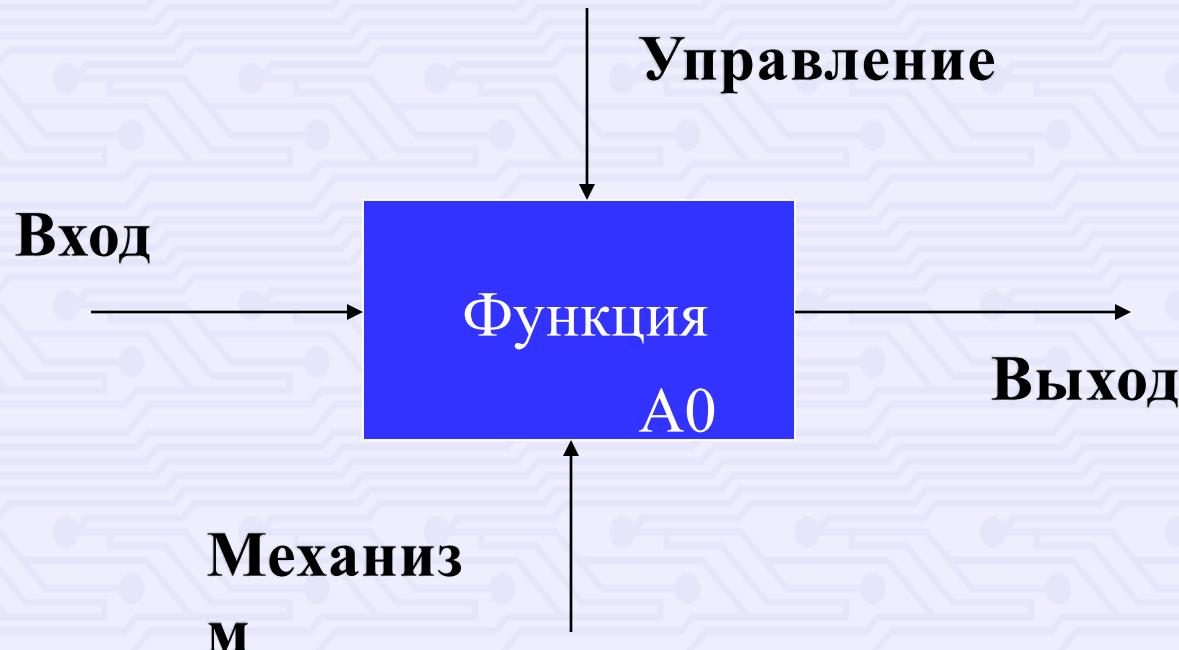
Функциональная модель SADT

Правила:

- уникальность меток и наименований;
- ограничение количества блоков на каждом уровне декомпозиции (3-6);
- синтаксические правила для графики;
- связность диаграмм;
- отделение организации от функции;
- разделение входов и управлений.

Состав функциональной модели

Результатом применения метода SADT является модель, которая состоит из диаграмм, фрагментов текстов и глоссария, имеющих ссылки друг на друга.



Конструктивные элементы диаграмм

- Функциональный блок (Activity Box):
 - ◆ изображается в виде прямоугольника;
 - ◆ представляет некоторую функцию, рассматриваемую в рамках системы;
 - ◆ название функционального блока должно иметь глагольное наклонение.
- Функциональные блоки на диаграммах декомпозиции обычно располагаются в порядке доминирования – по диагонали от левого верхнего угла к правому нижнему.

Конструктивные элементы диаграмм

- Интерфейсная дуга (Arrow):
 - ◆ изображается в виде стрелки;
 - ◆ обозначает элемент (объект или документ), который обрабатывается функциональным блоком или влияет на функциональный блок;
 - ◆ название дуги должно быть оборотом существительного.

Конструктивные элементы диаграмм

- *Вход* - это данные и объекты, потребляемые или изменяемые процессом.
- *Выход*- это основной результат процесса, конечный продукт (должна быть хотя бы одна дуга).
- *Управление* - стратегии и процедуры, которыми управляется процесс (должна быть хотя бы одна дуга).
- *Механизм*- ресурсы, исполняющие процесс (исполнитель).
- *Вызов* - это разновидность механизма, которая позволяет использовать одну и ту же часть диаграммы в нескольких моделях или в нескольких частях одной модели.

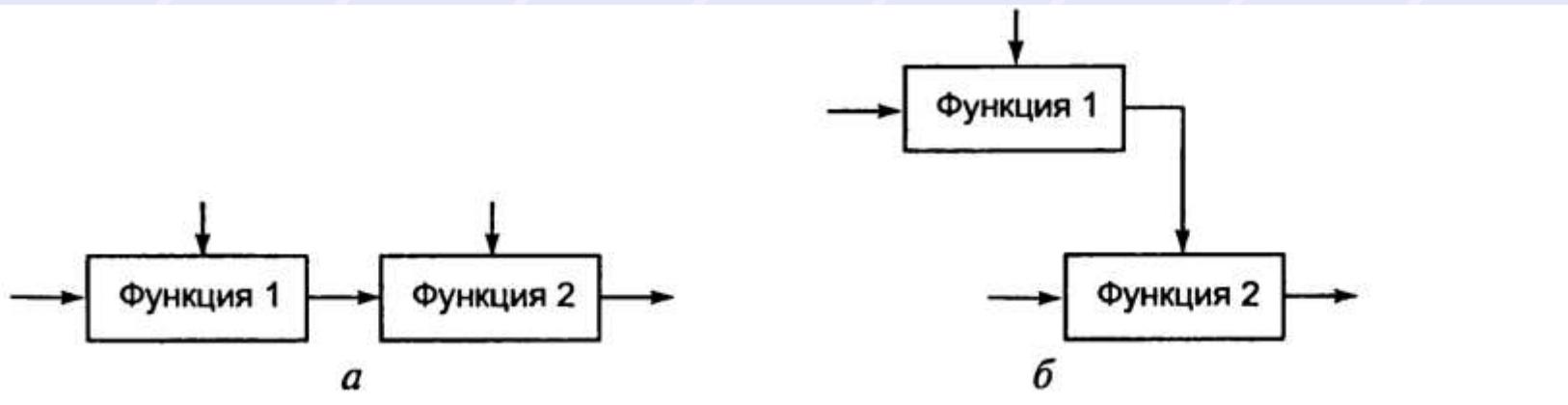
Конструктивные элементы диаграмм

- Каждый функциональный блок должен иметь *уникальный номер* и, по крайней мере, *одну управляющую и одну исходящую дуги*.
- Каждая интерфейсная дуга должна иметь *的独特ое название*.

Конструктивные элементы диаграмм

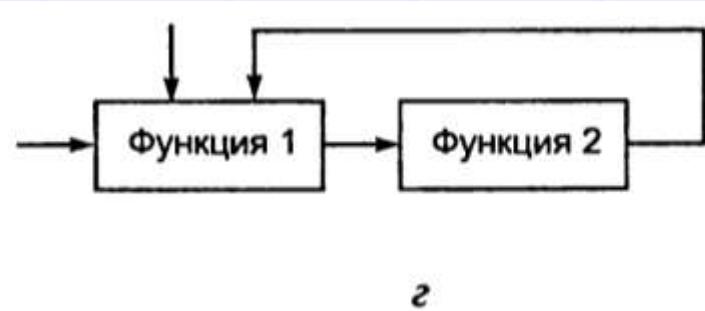
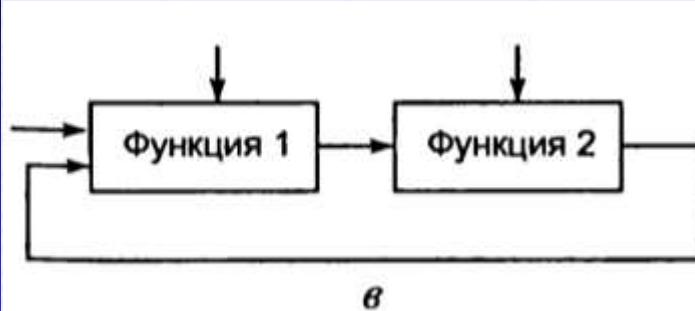
В функциональных диаграммах SADT различают пять типов влияний блоков друг на друга:

- вход-выход блока подается на вход блока с меньшим доминированием, т. е. следующего (а);
- управление. Выход блока используется как управление для блока с меньшим доминированием (б);



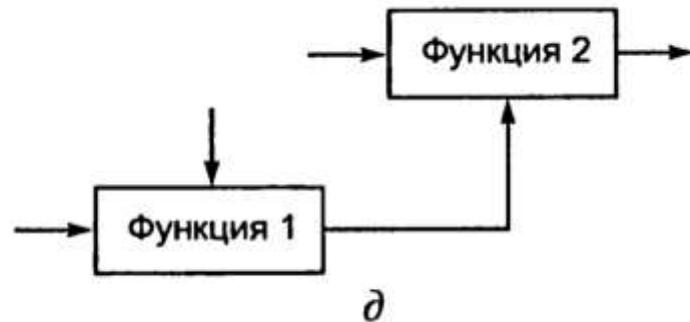
Конструктивные элементы диаграмм

- обратная связь по входу. Выход блока подается на вход блока с большим доминированием (в);
- обратная связь по управлению. Выход блока используется как управляющая информация для блока с большим доминированием (г);



Конструктивные элементы диаграмм

- выход-исполнитель. Выход блока используется как механизм для другого блока (д).



Конструктивные элементы диаграмм

Виды интерфейсных дуг

Граничные

Внутренние

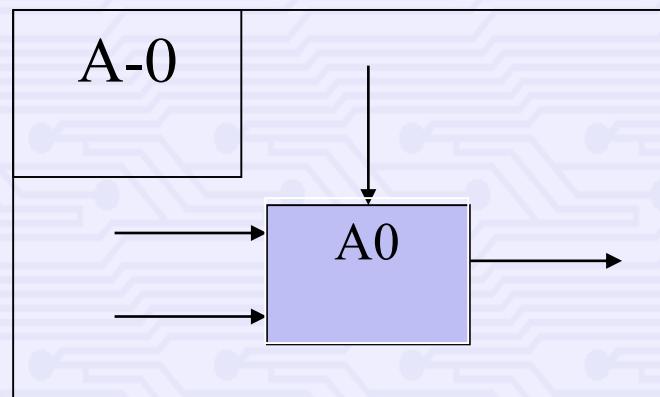
Связь с внешним
миром

Связь блоков между
собой

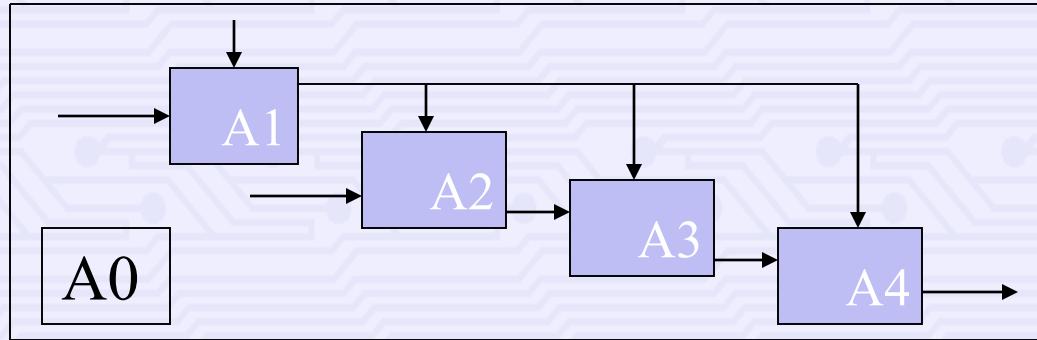
Состав функциональной модели

Диаграмма с одним функциональным блоком называется **контекстной диаграммой**, и обозначается идентификатором “A-0”.

Общее представление.

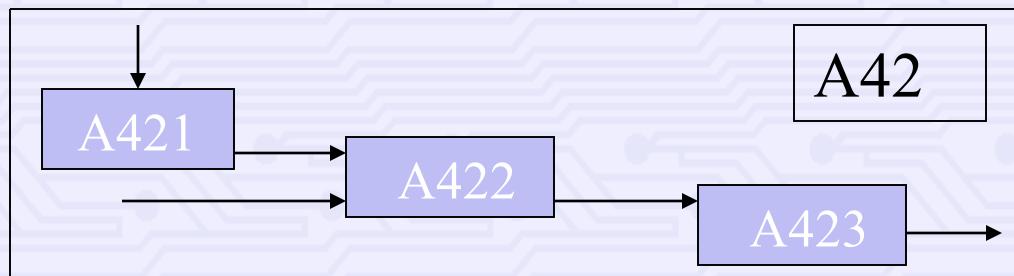
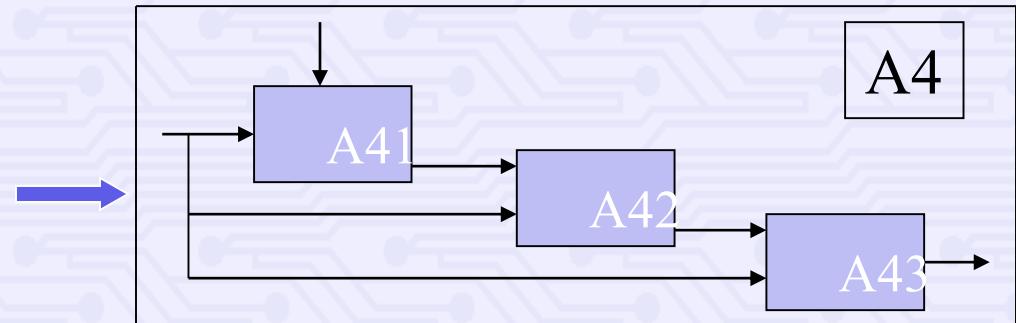


Декомпозиция диаграмм

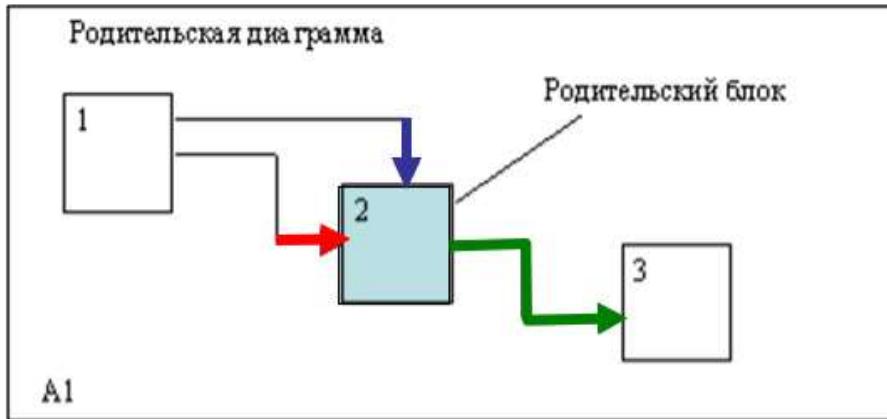


Более детальное представление.

Верхняя диаграмма является родительской для нижней диаграммы



Декомпозиция диаграмм



- Соответствие между дугами на родительской и на дочерней диаграммах должно быть полным и непротиворечивым



Декомпозиция диаграмм

Тоннельные дуги



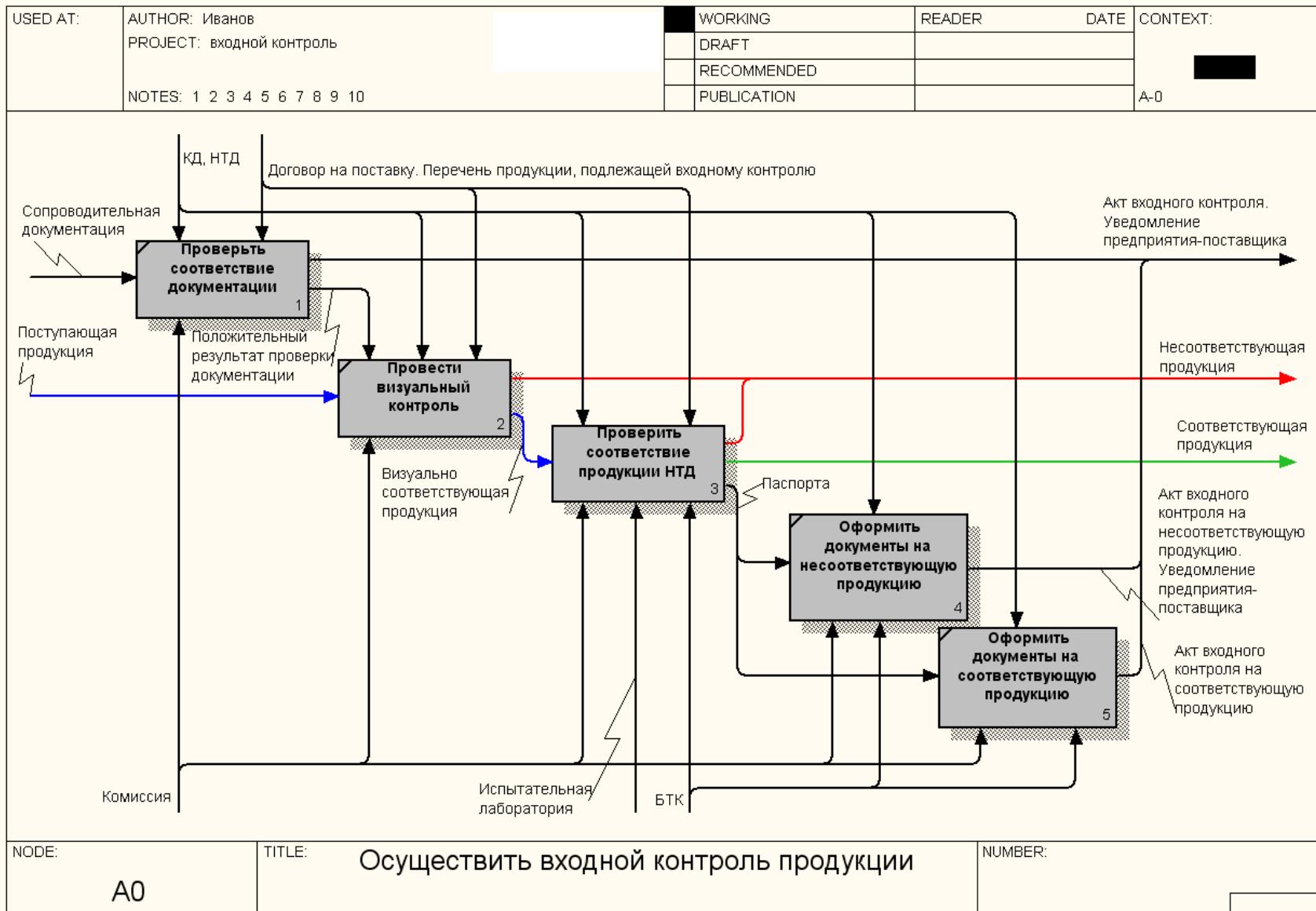
Вновь внесенные граничные стрелки на диаграмме декомпозиции нижнего уровня изображаются в квадратных скобках и автоматически не появляются на диаграмме верхнего уровня.

Декомпозиция диаграмм

Дуга "входит в тоннель", если:

- она является внешней дугой, которая отсутствует на родительской диаграмме (дуга имеет скрытый источник);
- она касается блока, но не появляется на диаграмме, которая его декомпозирует (имеет скрытый приемник).

Пример



Семейство методологий IDEF

- **IDEF0** – методология функционального моделирования производственной среды или систем, основана на SADT;
- **IDEF1** – методология информационного моделирования производственной среды или системы, основана на реляционной теории Кодда и использовании ER-диаграмм (диаграмм «Сущность – Связь»)
- **IDEF2** – методология динамического моделирования производственной среды или системы; отображает изменяющееся во времени поведение функций, информации и ресурсов системы

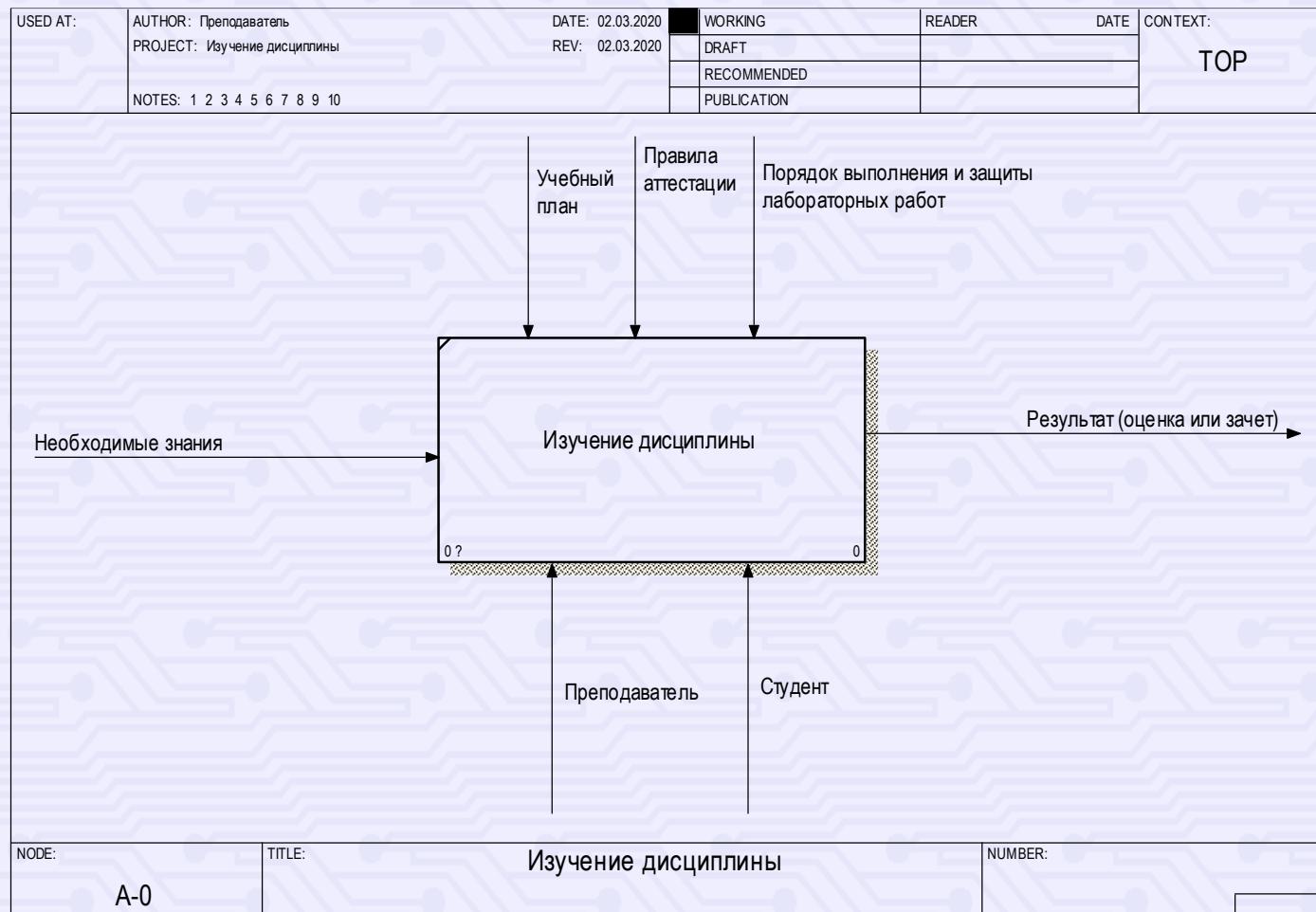
Семейство методологий IDEF

- IDEF1X – методология семантического моделирования данных, представляет собой расширение методологии IDEF1;
- IDEF3 – методология моделирования сценариев процессов, происходящих в производственной среде или системе; отображает состояния объектов и потоков данных, связи между ситуациями и событиями; используется для документирования процессов, происходящих в производственной среде или системе.

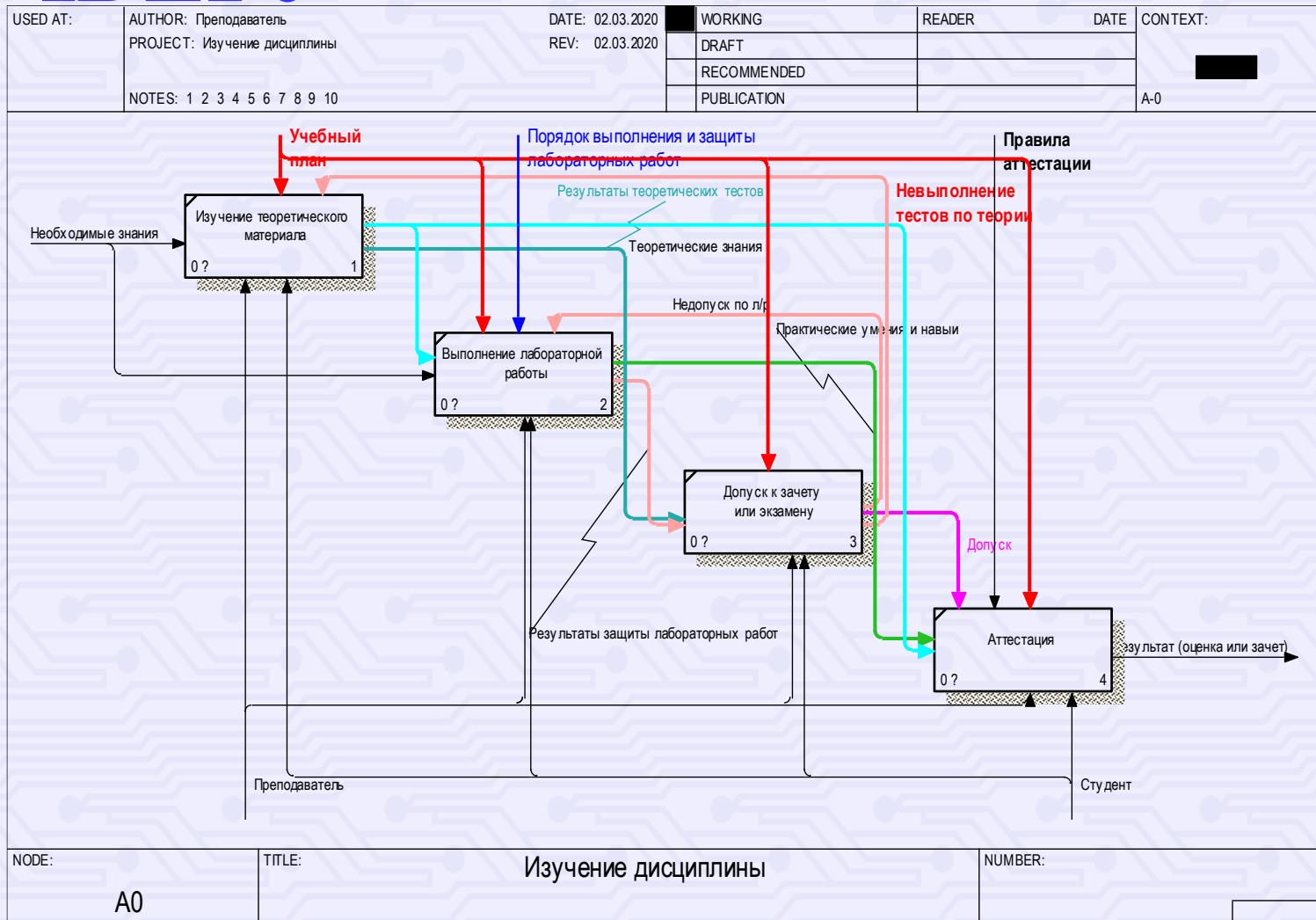
ПРИМЕР МОДЕЛИРОВАНИЯ IDEF0

- Изучение дисциплины в университете
- Цель: определить основные этапы изучения дисциплины и их влияние на конечный результат – оценку знаний студента преподавателем.
- Точка зрения: преподаватель

ПРИМЕР МОДЕЛИРОВАНИЯ IDEFO



ПРИМЕР МОДЕЛИРОВАНИЯ IDEFO



План лекции:

- 3. Диаграммы потоков данных**
- 4. Моделирование логики взаимодействия процессов**
- 5. Спецификация требований (техническое задание)**

Семейство методологий IDEF

- **IDEF0** – методология функционального моделирования производственной среды или систем, основана на SADT;
- **IDEF1** – методология информационного моделирования производственной среды или системы, основана на реляционной теории Кодда и использовании ER-диаграмм (диаграмм «Сущность – Связь»)
- **IDEF2** – методология динамического моделирования производственной среды или системы; отображает изменяющееся во времени поведение функций, информации и ресурсов системы

Семейство методологий IDEF

- IDEF1X – методология семантического моделирования данных, представляет собой расширение методологии IDEF1;
- IDEF3 – методология моделирования сценариев процессов, происходящих в производственной среде или системе; отображает состояния объектов и потоков данных, связи между ситуациями и событиями; используется для документирования процессов, происходящих в производственной среде или системе.

3 ДИАГРАММЫ ПОТОКОВ ДАННЫХ



Диаграммы потоков данных

Data Flow Diagrams (DFD) представляют собой иерархию функциональных процессов, связанных потоками данных.

Цель такого представления - продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.

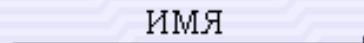
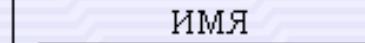
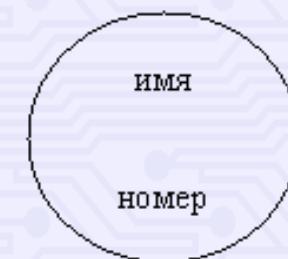
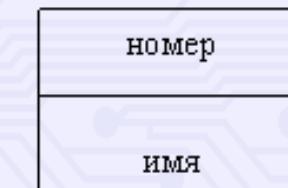
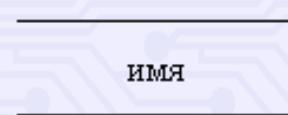
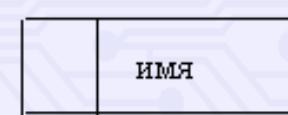
Диаграммы потоков данных

- Диаграммы верхних уровней иерархии (контекстные диаграммы) определяют основные процессы или подсистемы ПС с внешними входами и выходами.
- Они детализируются при помощи диаграмм нижнего уровня. Такая декомпозиция продолжается, создавая многоуровневую иерархию диаграмм, до тех пор, пока не будет достигнут такой уровень декомпозиции, на котором процесс становится элементарным и детализировать его далее невозможно.

Диаграммы потоков данных

- Основными компонентами диаграмм потоков данных являются:
 - ◆ внешние сущности;
 - ◆ процессы;
 - ◆ накопители (хранилище) данных;
 - ◆ потоки данных.
- Для изображения DFD традиционно используются две различные нотации:
 - ◆ Йодана (Yourdon)
 - ◆ Гейна-Сарсона (Gane-Sarson).

Диаграммы потоков данных

Компонента	Нотация Йодана	Нотация Гейна-Сарсона
ПОТОК ДАННЫХ		
ПРОЦЕСС		
ХРАНИЛИЩЕ		
ВНЕШНЯЯ СУЩНОСТЬ		

Диаграммы потоков данных

- **Внешняя сущность** - материальный объект или физическое лицо, выступающие в качестве источников или приемников информации, например, заказчики, персонал, поставщики, инструкции, банк и т. п.

Внешние сущности, как правило, отображаются только на контекстной диаграмме DFD. В процессе анализа и проектирования некоторые внешние сущности могут быть перенесены на диаграммы декомпозиции, если это необходимо, или, наоборот, часть процессов (подсистем) может быть представлена как внешняя сущность.

Диаграммы потоков данных

- *Процесс* - преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом.

На верхних уровнях иерархии, когда процессы еще не определены, вместо понятия «процесс» используют понятия «система» и «подсистема», которые обозначают соответственно систему в целом или ее функционально законченную часть.

Диаграммы потоков данных

- *Хранилище данных* - абстрактное устройство для хранения информации. Тип устройства и способы помещения, извлечения и хранения для такого устройства не детализируют.

Физически это может быть база данных, файл, таблица в оперативной памяти, картотека на бумаге и т. п.

Диаграммы потоков данных

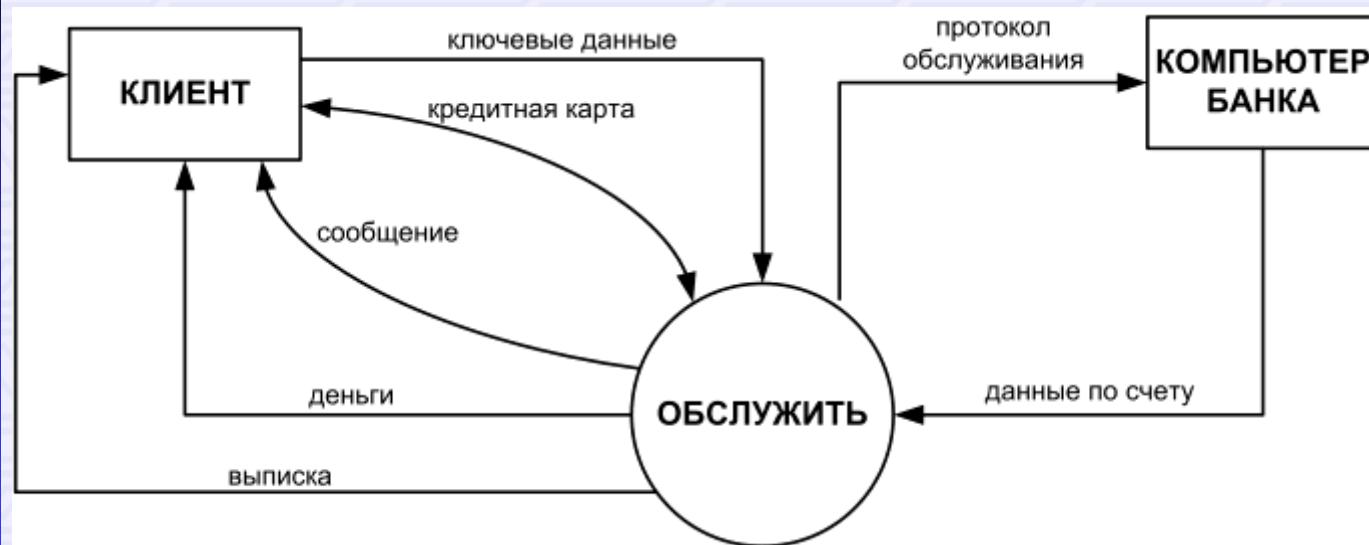
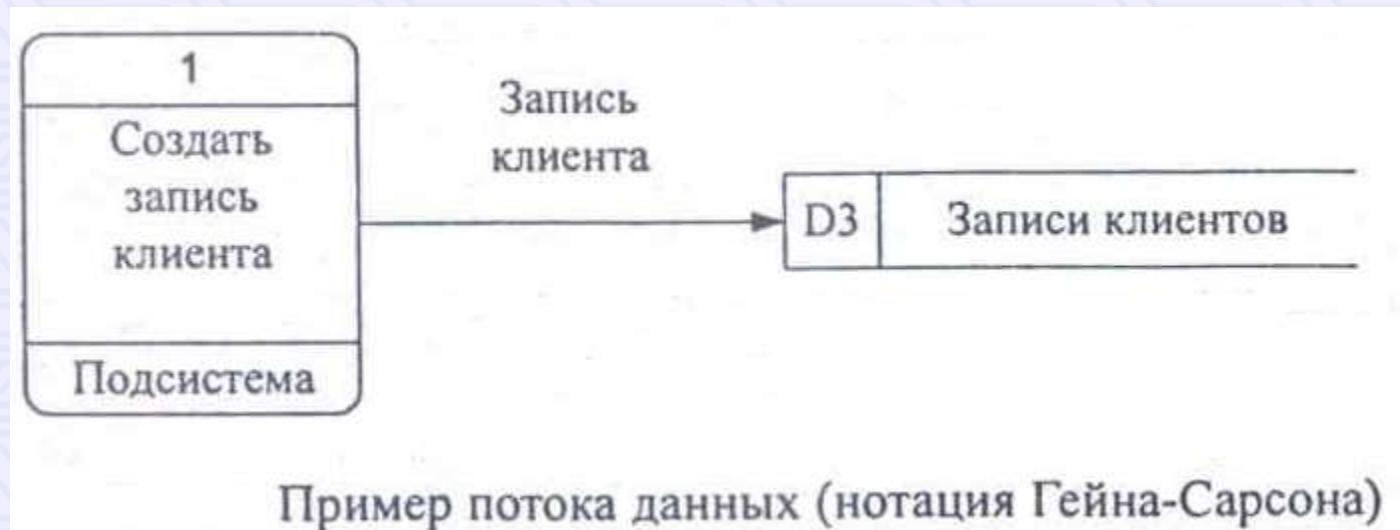
- **Поток данных** определяет информацию (материальный объект), передаваемую через некоторое соединение от источника к приемнику.

Каждый поток данных имеет имя, отражающее его содержание.

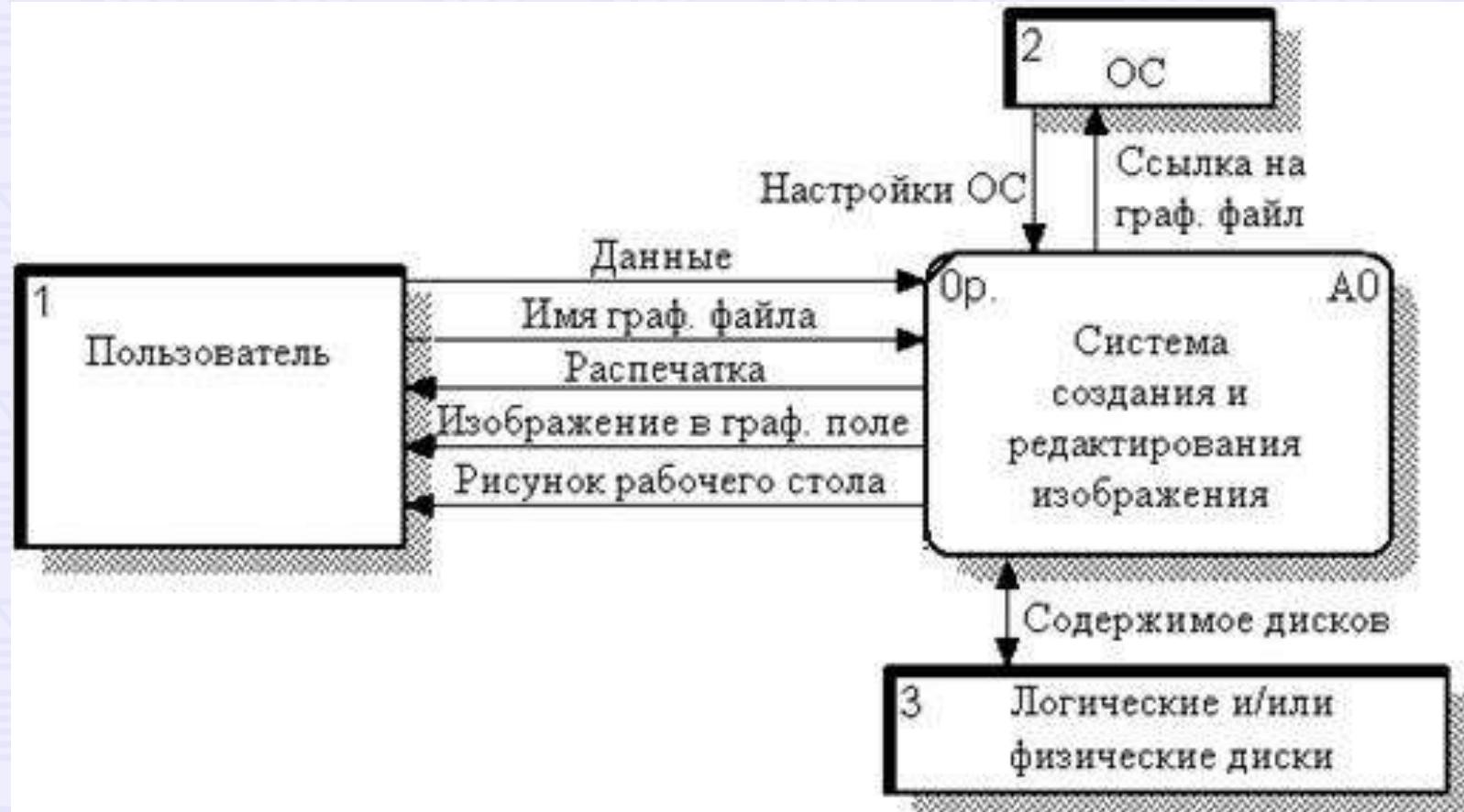
Направление стрелки показывает направление потока данных.

Физически процесс передачи информации может происходить по кабелям под управлением программы или программной системы или вручную при участии устройств или людей вне проектируемой системы.

Примеры



Примеры



Правила детализации контекстных диаграмм:

- **правило балансировки.** Означает, что при детализации подсистемы можно использовать только те компоненты (подсистемы, процессы, внешние сущности, накопители данных), с которыми она имеет информационную связь на родительской диаграмме;
- **правило нумерации.** Означает, что при детализации подсистем должна поддерживаться их иерархическая нумерация. Например, подсистемы, детализирующие подсистему с номером 2, получают номера 2.1, 2.2, 2.3 и т. д.

Правила детализации контекстных диаграмм:

Для достижения этого целесообразно:

- размещать на каждой диаграмме от 3 до 6-7 процессов;
- не загромождать диаграммы не существенными на данном уровне деталями;
- декомпозицию потоков данных осуществлять *параллельно* с декомпозицией процессов;
- выбирать ясные, отражающие суть дела имена процессов и потоков.

Правила детализации контекстных диаграмм:

Каждый процесс может быть детализирован при помощи **DFD** или (если процесс элементарный) **спецификации**.

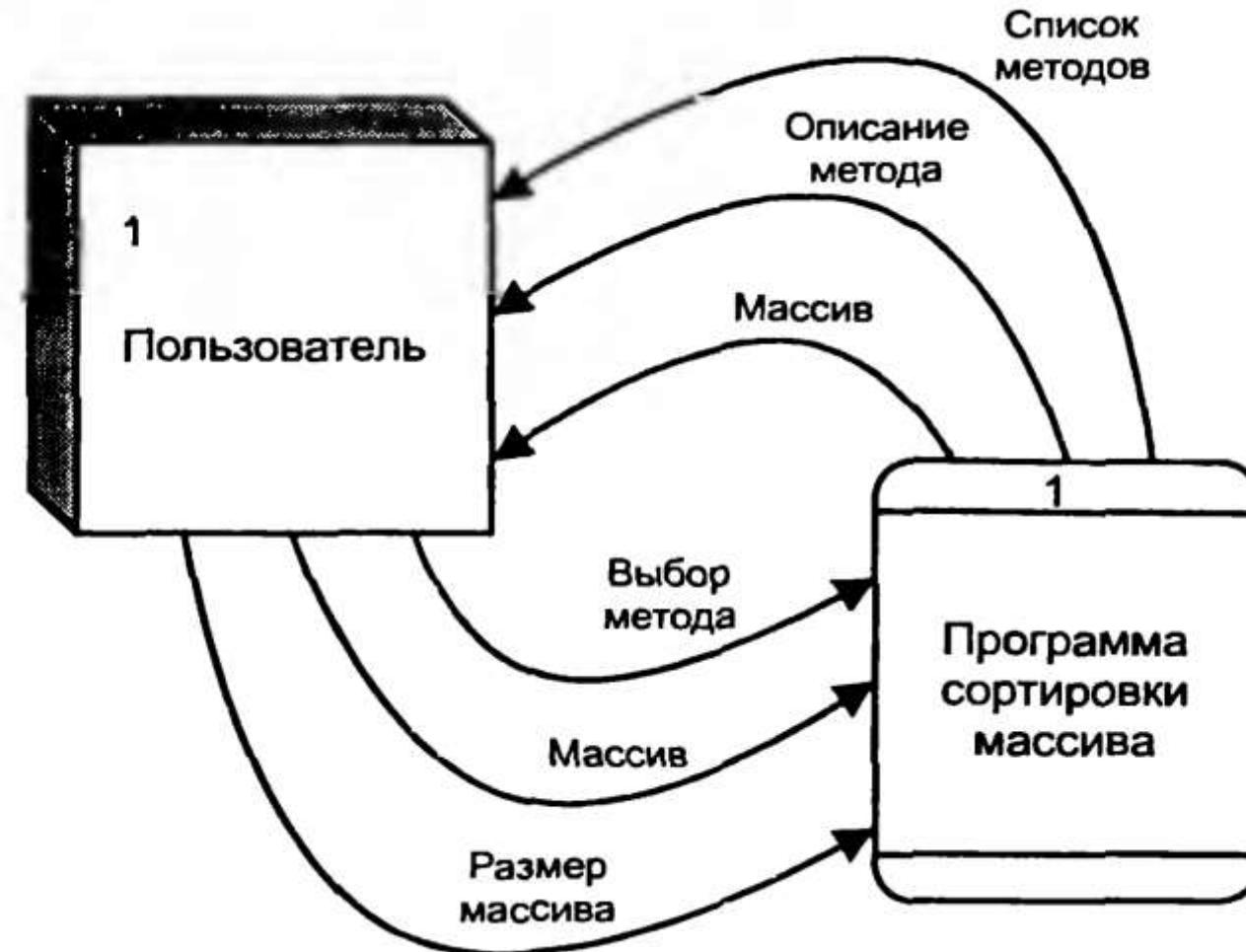
Спецификация процесса должна формулировать его основные функции таким образом, чтобы в дальнейшем специалист, выполняющий реализацию проекта, смог выполнить их или разработать соответствующую программу.

Спецификация является конечной вершиной иерархии.
Спецификация – это описание процесса (операции) на естественном языке, псевдокоде, в виде блок-схемы алгоритма, таблиц, математических формул и т.д.

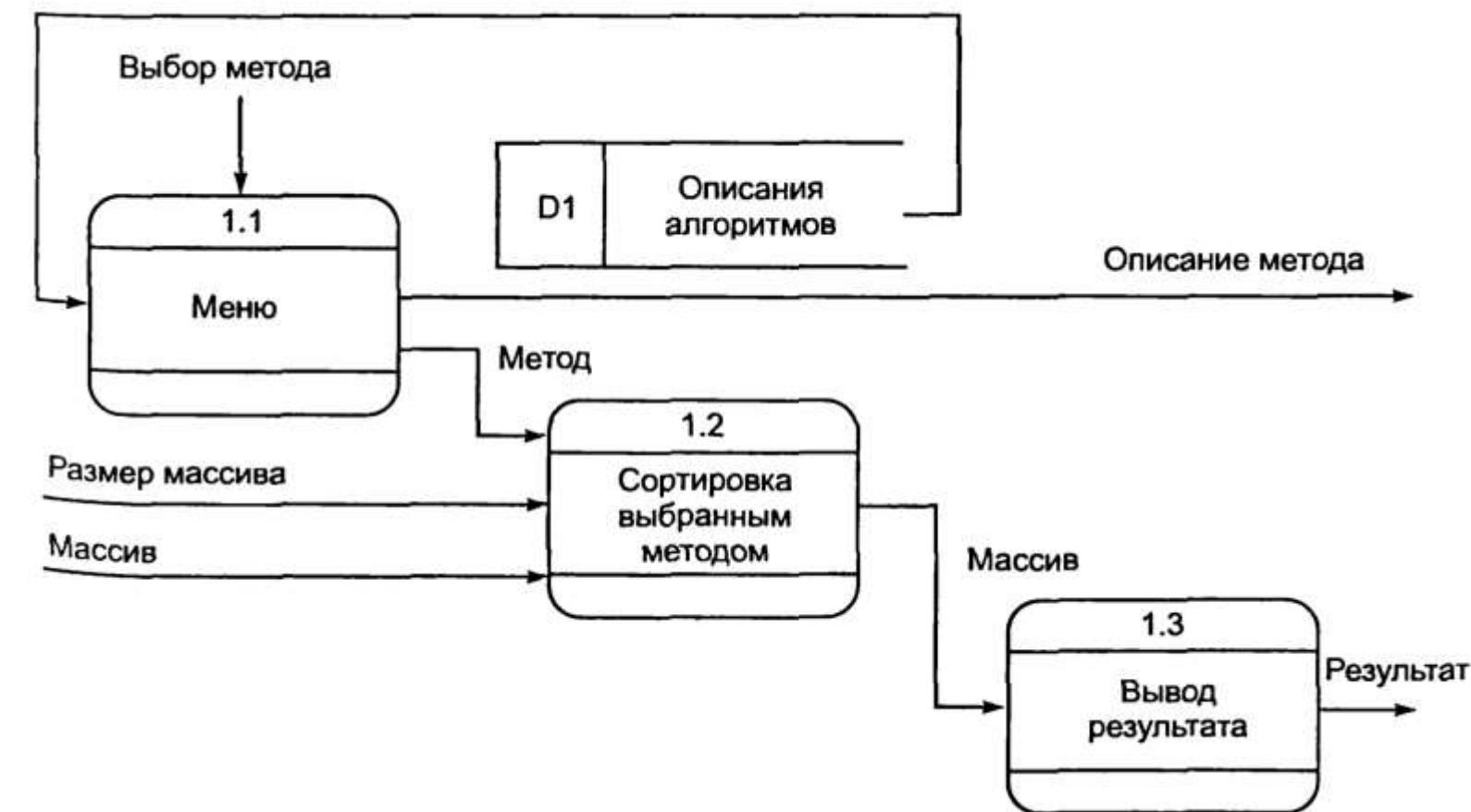
Правила построения диаграмм:

- Каждый процесс должен иметь хотя бы один вход и один выход.
- Стрелки не могут связывать напрямую хранилища данных, все связи идут через процессы.
- Все процессы должны быть связаны либо с другими процессами, либо с другими хранилищами данных.

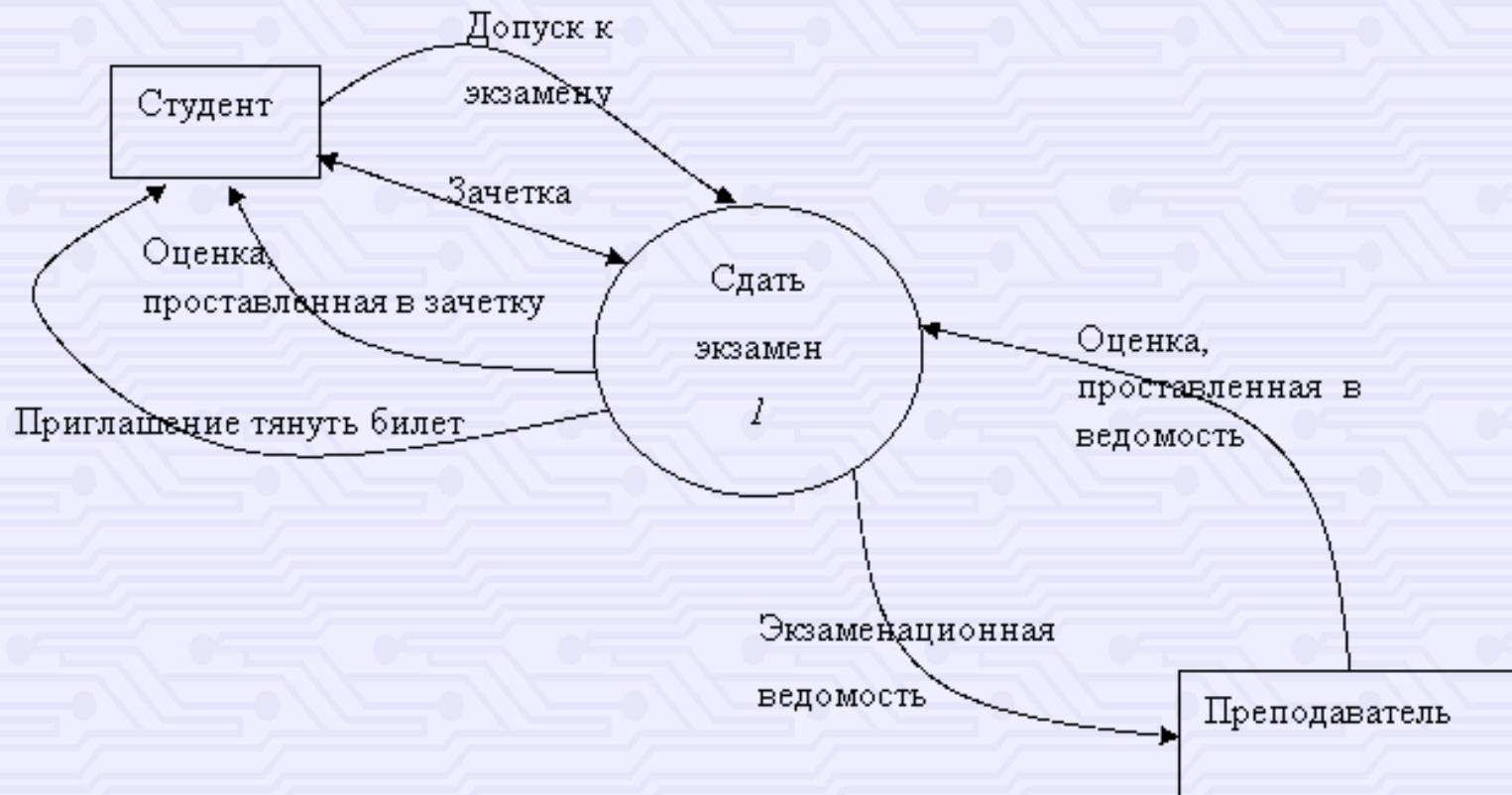
Пример в нотации Гейна-Сарсона



Пример в нотации Гейна-Сарсона



Пример в нотации Йодана



Пример в нотации Йодана



Этапы процесса построения модели:

1. Выделение множества требований в основные функциональные группы — процессы.
2. Выявление внешних объектов, связанных с разрабатываемой системой.
3. Идентификация основных потоков информации, циркулирующей между системой и внешними объектами.
4. Предварительная разработка контекстной диаграммы.
5. Проверка предварительной контекстной диаграммы и внесение в нее изменений.

Этапы процесса построения модели:

6. Построение контекстной диаграммы путем объединения всех процессов предварительной диаграммы в один процесс, а также группирования потоков.
7. Проверка основных требований контекстной диаграммы.
8. Декомпозиция каждого процесса текущей DFD с помощью детализирующей диаграммы или спецификации процесса.
9. Проверка основных требований по DFD соответствующего уровня.

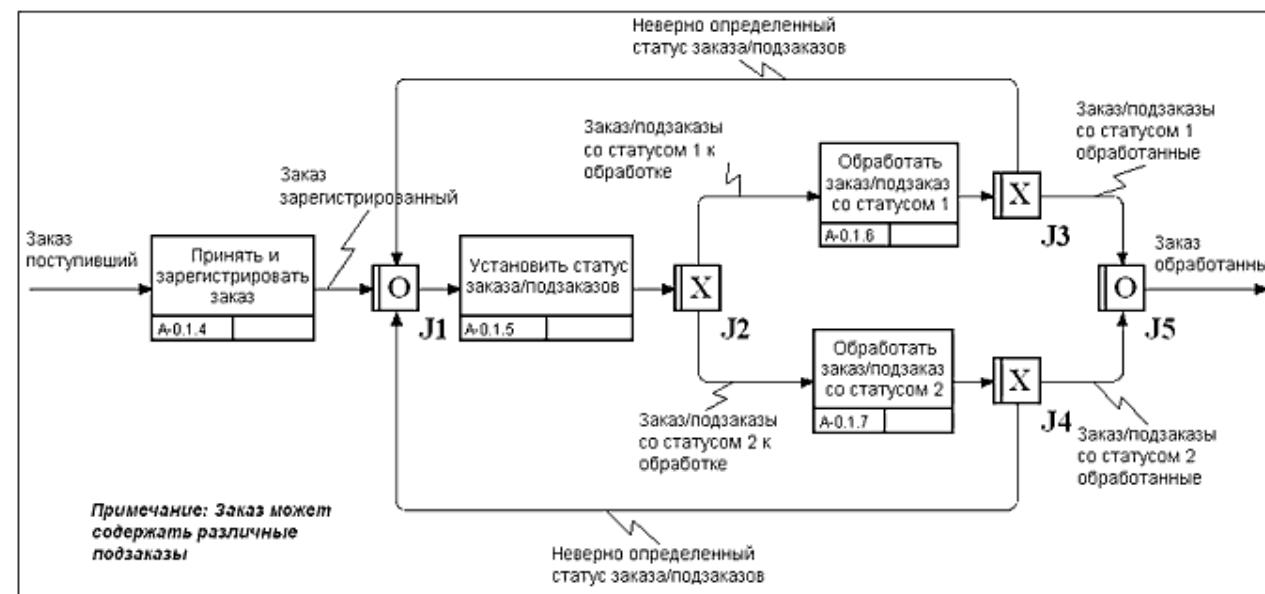
Этапы процесса построения модели:

10. Добавление определений новых потоков в словарь данных при каждом их появлении на диаграммах.
11. Проверка полноты и наглядности модели после построения каждого двух-трех уровней.

Словарь данных представляет собой определенным образом организованный список всех элементов данных системы с их точными определениями.

Спецификация процесса (СП) используется для описания функционирования процесса в случае отсутствия необходимости детализировать его с помощью DFD.

3 МОДЕЛИРОВАНИЕ ЛОГИКИ ВЗАИМОДЕЙСТВИЯ ПРОЦЕССОВ



Пример IDEF3-диаграммы

Процессная модель IDEF3

В общем случае, процесс – это упорядоченная последовательность действий.

Следовательно, процессная модель IDEF3 позволяет:

- Отразить последовательность процессов
- Показать логику взаимодействия элементов системы.

Цель IDEF3 - дать возможность аналитикам описать ситуацию, когда процессы выполняются в определенной последовательности, а также объекты, участвующие совместно в одном процессе.

Основные компоненты IDEF3-модели

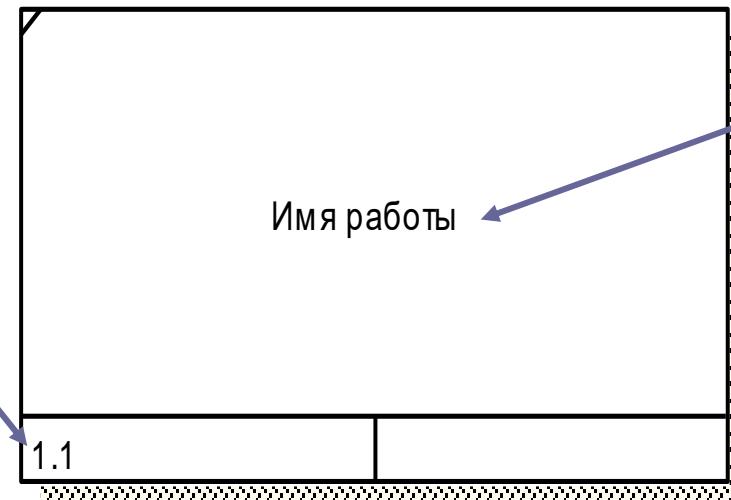
- 1) Единицы работ
- 2) Связи
- 3) Перекрестки
- 4) Объекты ссылок.

Единицы работ

■ Единица работ (UOW, Unit of Work)

является центральным компонентом модели.

Номер работы является уникальным, присваивается при ее создании и не меняется никогда



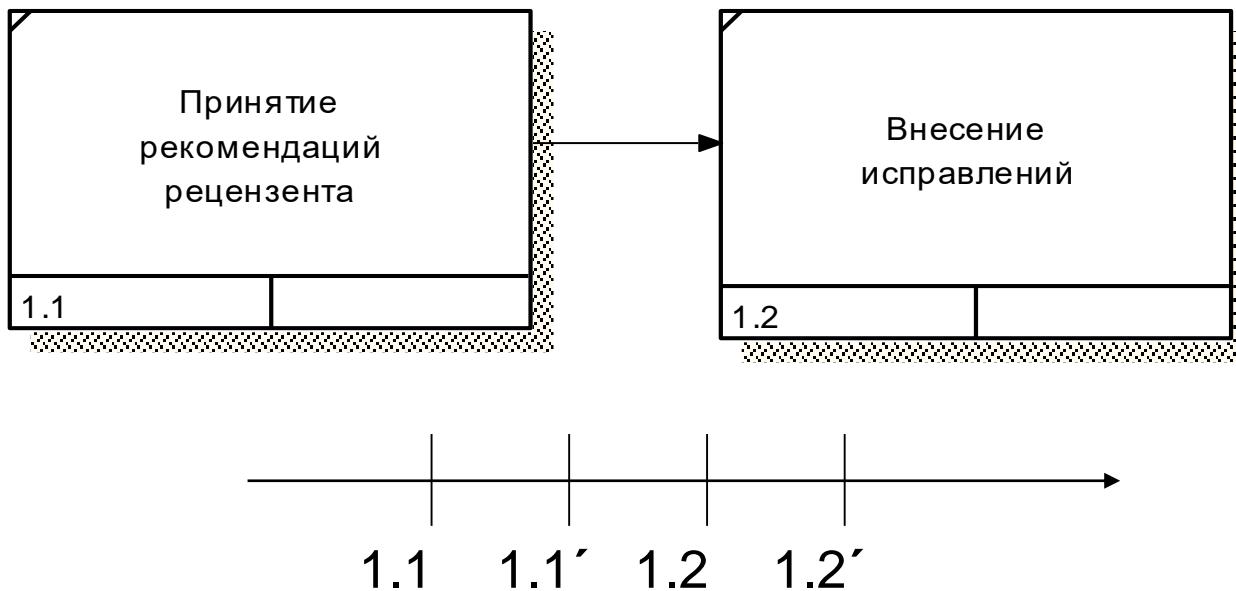
Словосочетание с отглагольным существительным, изображающим действие (выполнение, изготовление,...)
Или
Инфинитив глагола (изготовить продукцию)

Связи

- **Связи** показывают взаимоотношения работ.
- Связи *однонаправлены* и могут быть направлены куда угодно
- Обычно диаграммы рисуют таким образом, чтобы связи были направлены слева направо
- Различают *3 типа* связей:
 - Старшая стрелка
 - Стрелка отношений
 - Поток объектов.

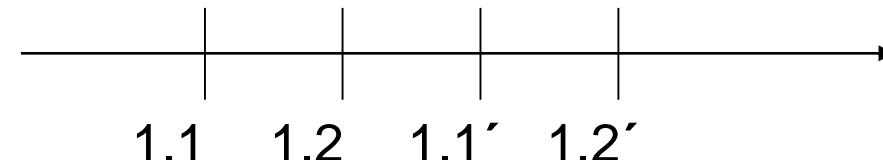
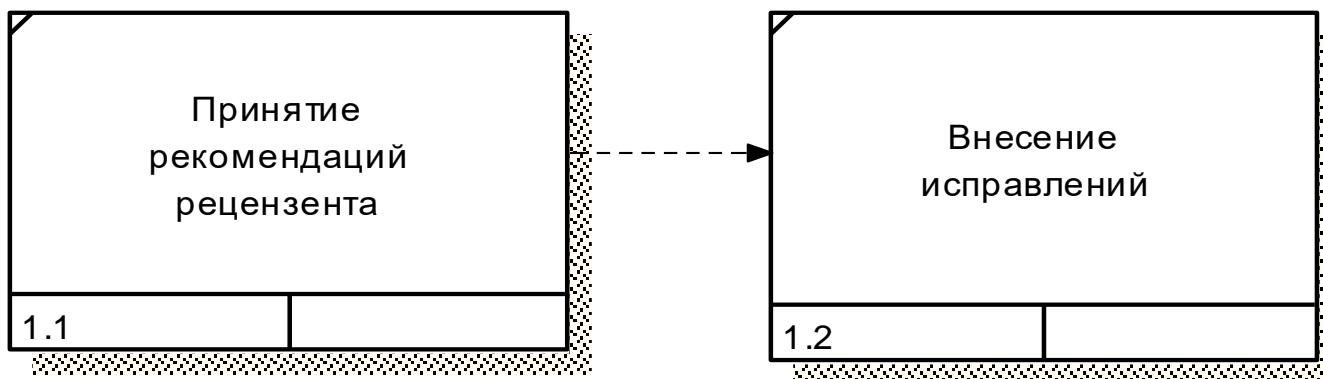
Связь «старшая стрелка»

- Связь типа «*временное предшествование*» - *Precedence*
- Соединяет единицы работ
- Показывает, что работа-источник должна быть закончена *прежде*, чем начнется работа-цель



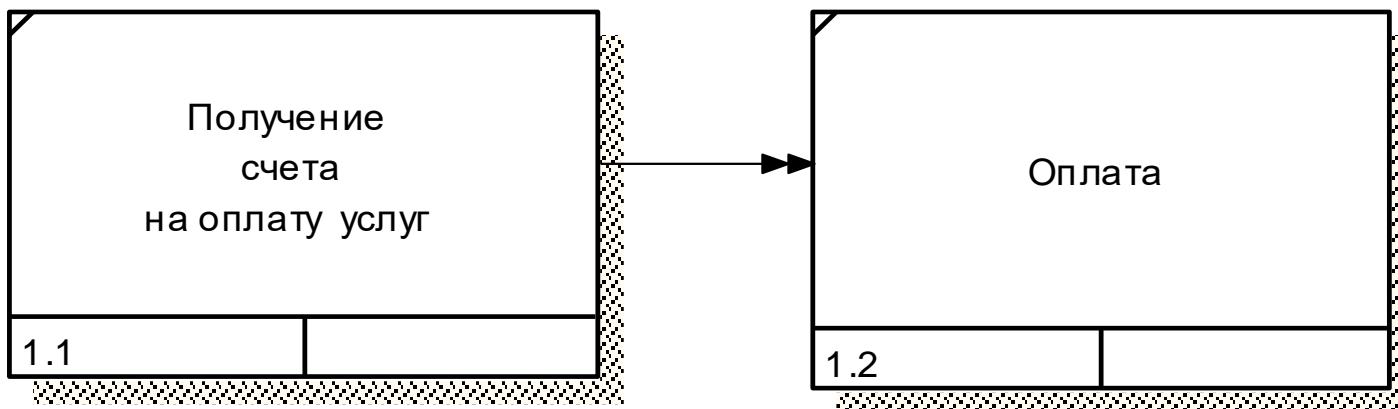
Стрелка отношений

- Связь типа **нечеткое отношение** - *Relational*
- Изображается в виде пунктирной линии, используется для изображения связи между единицами работ, а также между единицами работ и объектами ссылок



Поток объектов

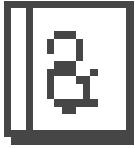
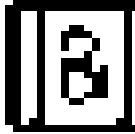
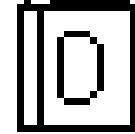
- Стрелка, изображающая поток объектов - *Object Flow*
- Применяется для описания того факта, что объект используется в двух и более единицах работ, например, когда объект порождается в одной работе и используется в другой



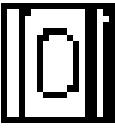
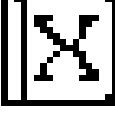
Перекрестки (соединения)

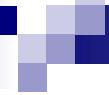
- Используются для отображения **логики взаимодействия** стрелок при их *слиянии* или *разветвлении*, для отображения множества событий, которые могут или должны быть завершены перед началом следующей работы.
- Различают перекрестки для *слияния* и *разветвления* стрелок.
- Перекрестки не могут быть одновременно использованы для слияния и разветвления стрелок.
- Все перекрестки на диаграммах нумеруются, каждый номер имеет префикс **J**.
- В отличие от других методологий (IDEF0, DFD) стрелки могут сливаться или разветвляться только через перекрестки.

Типы перекрестков

Обозна- чение	Наименов ание	Смысл в случае слияния стрелок	Смысл в случае разветвления стрелок
	Асинхрон- ное «И»	Все предшествующие процессы должны быть завершены	Все последующие процессы должны быть запущены
	Синхрон- ное «И»	Все предшествующие процессы должны быть завершены одновременно	Все последующие процессы запускаются одновременно
	Асинхрон- ное «ИЛИ»	Один или несколько предшествующих процессов должны быть завершены	Один или несколько следующих процессов должны быть запущены

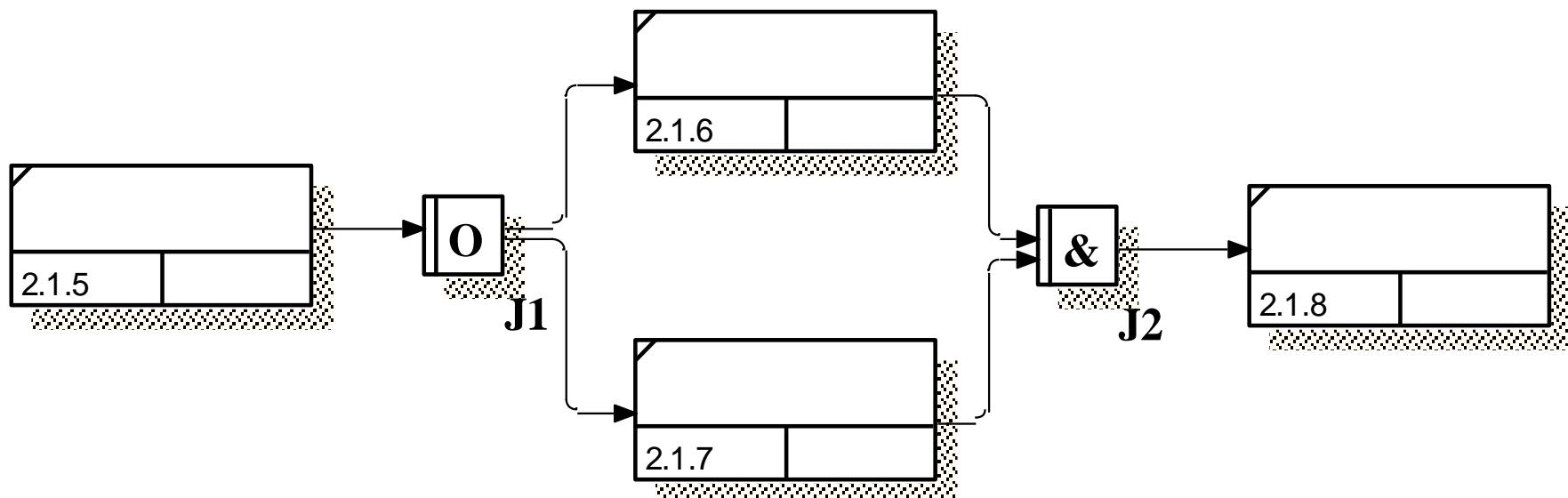
Типы перекрестков

Обозна- чение	Наименов- ание	Смысл в случае слияния стрелок	Смысл в случае разветвления стрелок
	Синхронн ое «ИЛИ»	Один или несколько предшествующих процессов должны быть завершены одновременно	Один или несколько следующих процессов должны быть запущены одновременно
	Эксклюзи вное (исключа ющее) «ИЛИ»	Только один предшествующий процесс должен быть завершен	Только один следующий процесс запускается



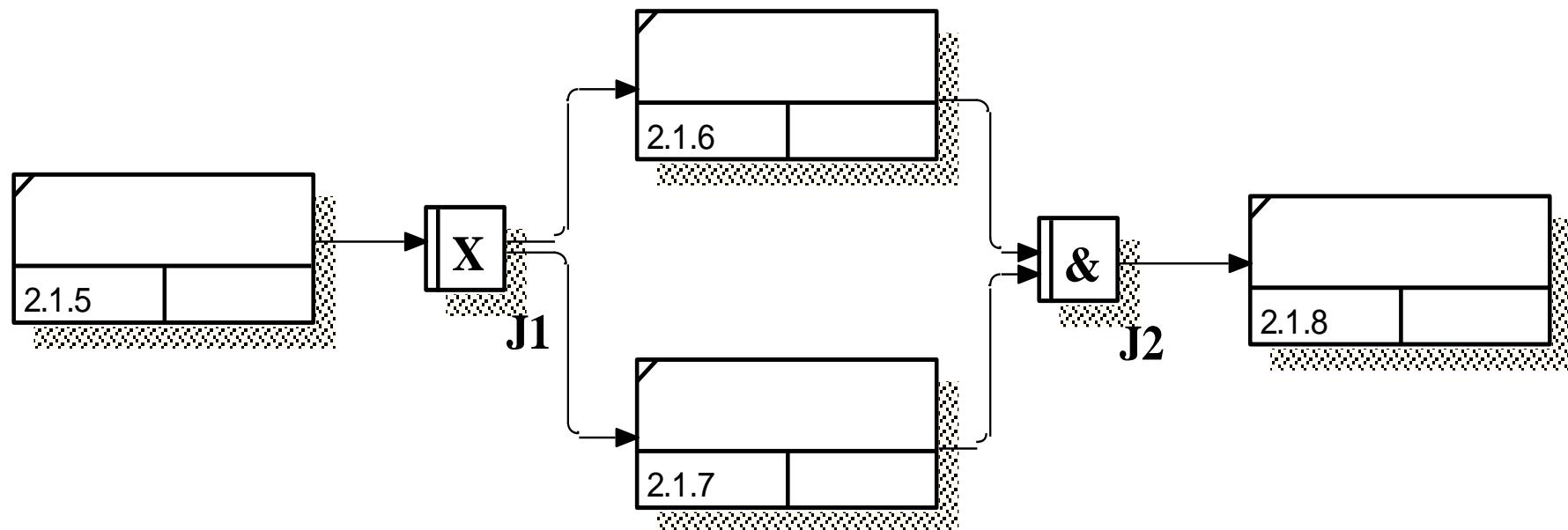
Правила создания перекрестков

1. Каждому перекрестку для слияния должен предшествовать перекресток для разветвления.
2. Перекресток для слияния «И» не может следовать за перекрестком для разветвления типа синхронного или асинхронного «ИЛИ»



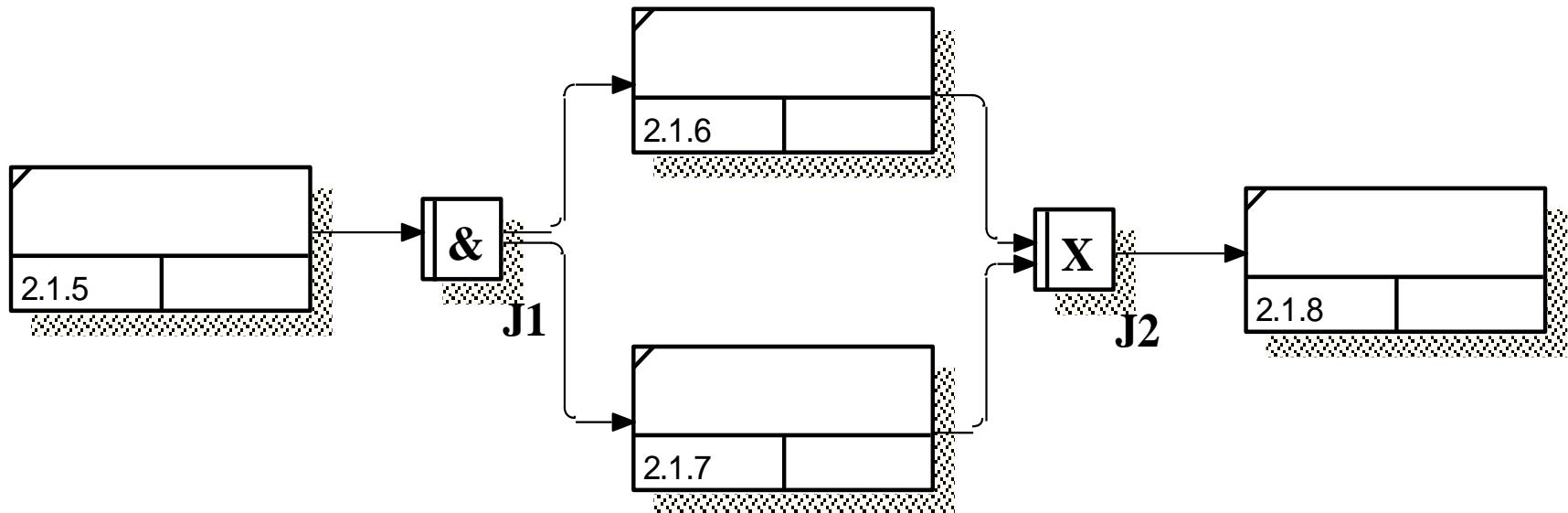
Правила создания перекрестков

3. Перекресток для слияния «И» не может следовать за перекрестком типа исключающего «ИЛИ»



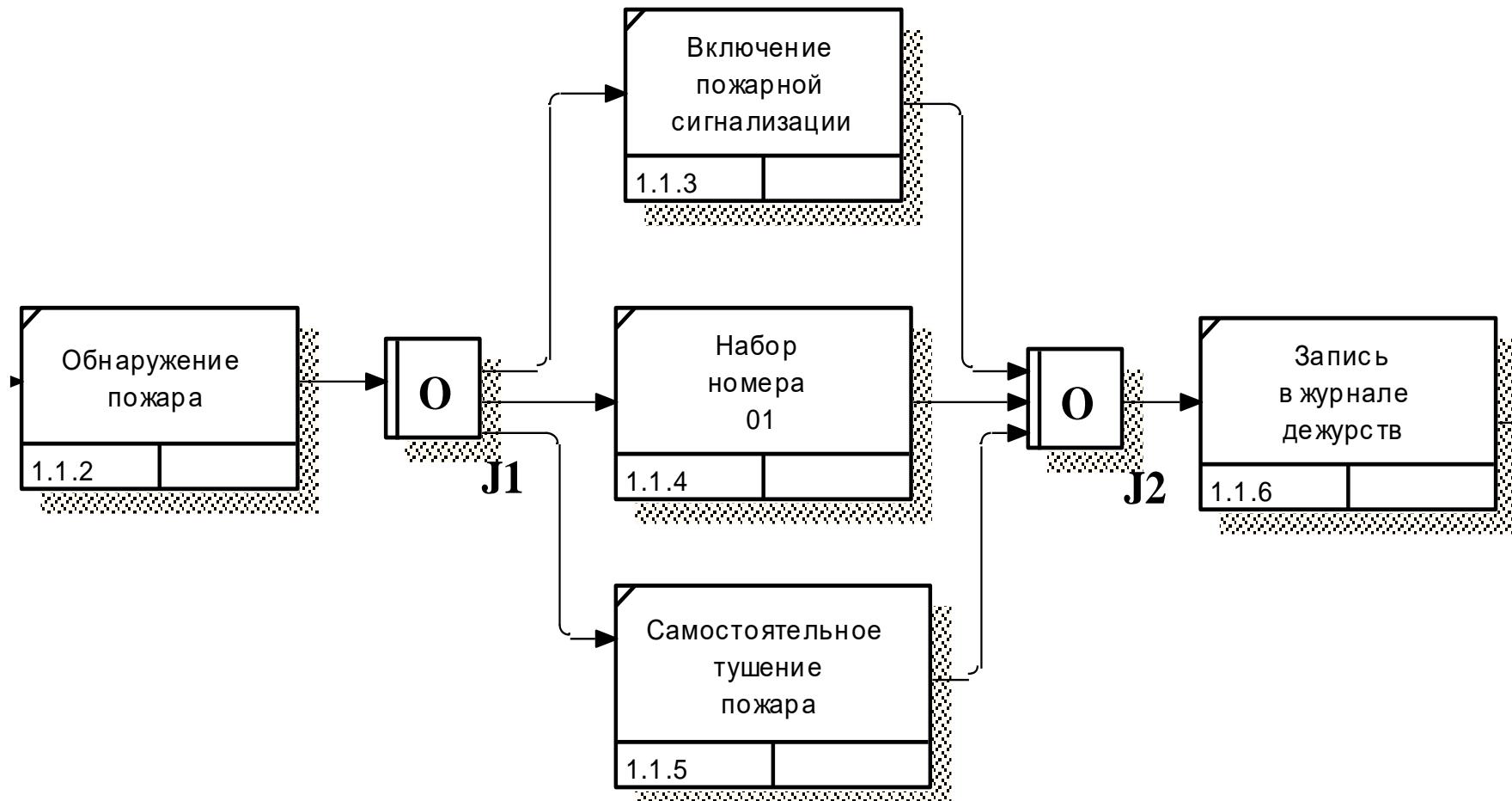
Правила создания перекрестков

4. Перекресток для слияния типа исключающего «ИЛИ» не может следовать за перекрестком для разветвления типа «И»



5. Перекресток, имеющий одну стрелку на одной стороне, должен иметь более одной стрелки на другой.

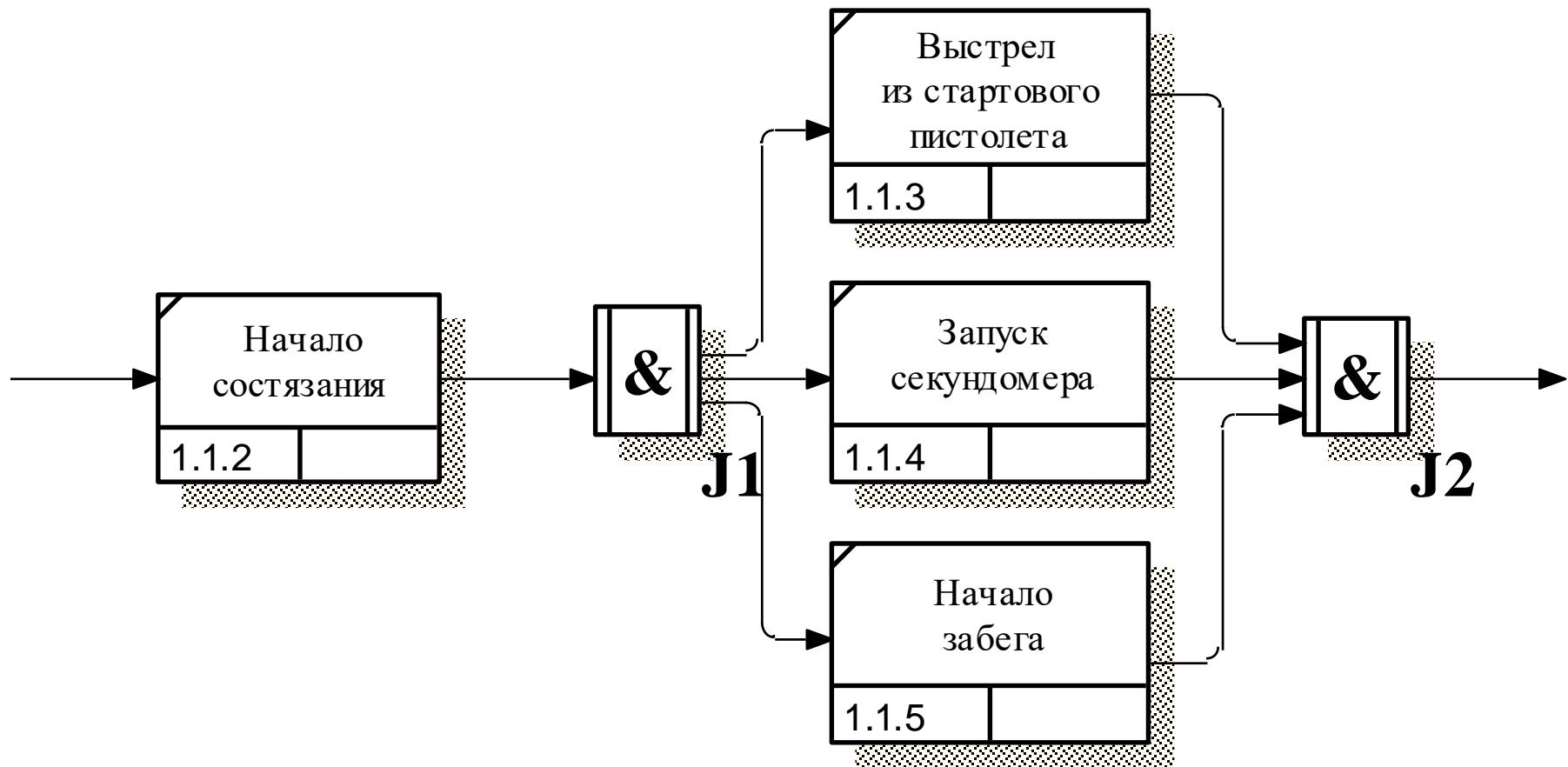
Примеры



Примеры

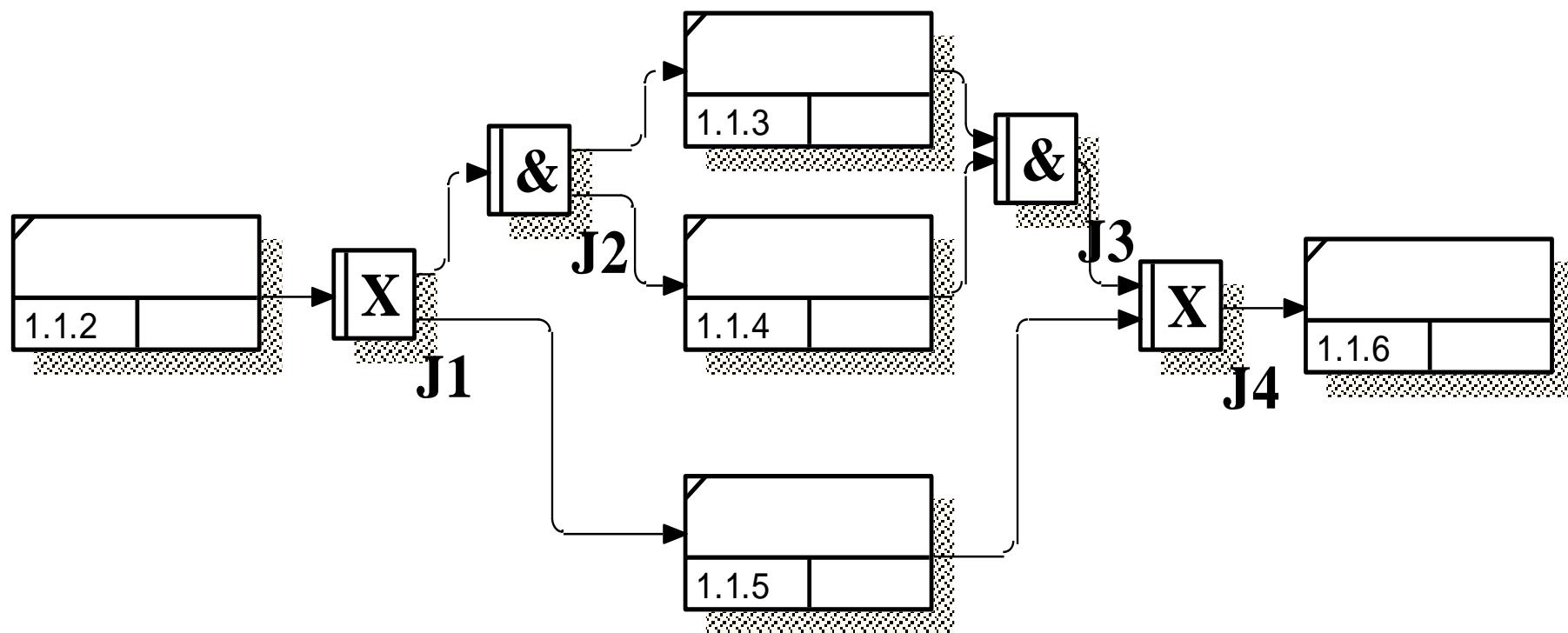


Примеры



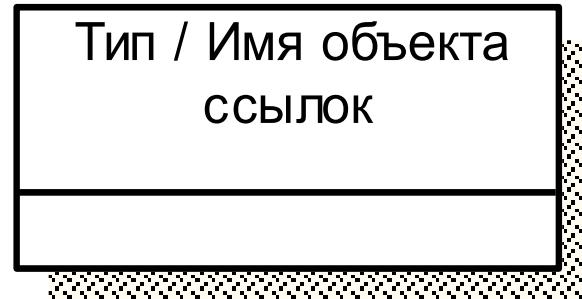
Комбинации перекрестков

- Перекрестки могут комбинироваться для создания сложных соединений



Объект ссылок

- выражает **идею, концепцию данных**, которые нельзя связать со стрелкой, перекрестком, работой
- используется при построении диаграммы для привлечения внимания пользователя к каким-либо важным аспектам модели



Объект ссылок

- Официальная спецификация IDEF3 различает 3 стиля объектов ссылок – безусловные (unconditional), синхронные (synchronous), асинхронные (asynchronous).
- BPWin поддерживает только безусловные объекты ссылок.

Типы объектов ссылок

Тип объекта ссылок	Назначение
1. <i>Object</i>	Используется для описания того, что в действии принимает участие какой-либо заслуживающий отдельного внимания объект
2. Ссылка <i>GOTO</i>	Используется для реализации цикличности выполнения действий. Этот объект также может относиться к перекрестку
3. Единица действий <i>UOB</i> (Unit of Behavior)	Используется для многократного отображения на диаграмме одного и того же действия, но без цикла

Типы объектов ссылок

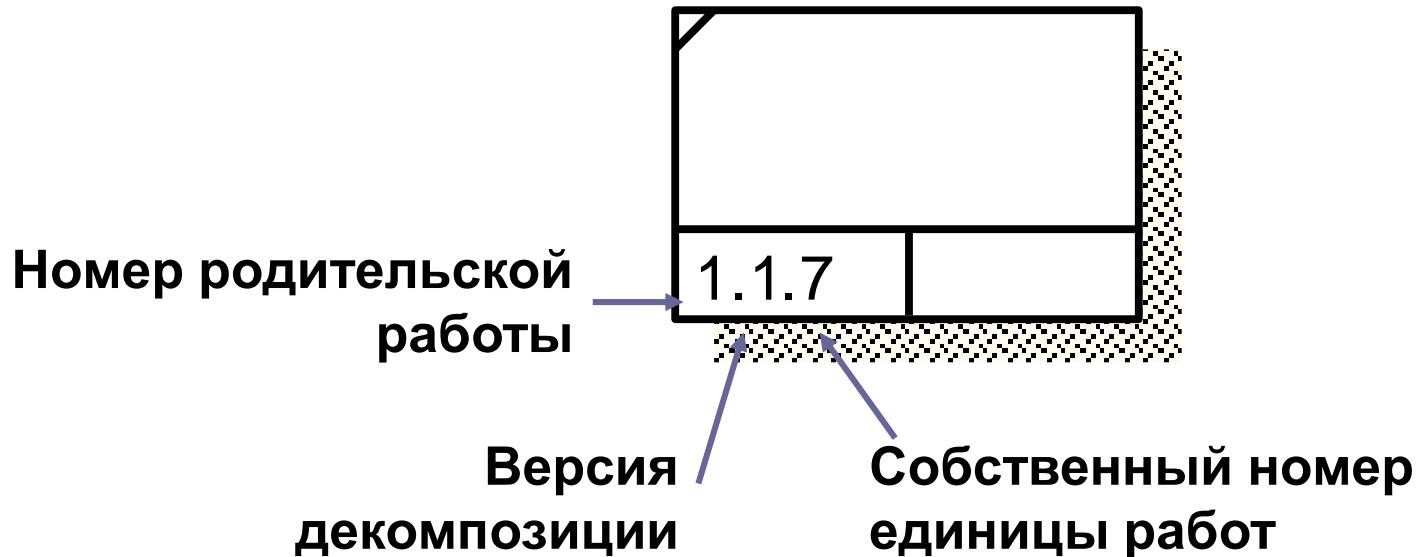
Тип объекта ссылок	Назначение
4. Заметка <i>(Note)</i>	Используется для документирования какой-либо важной информации общего характера, относящейся к изображаемому на диаграммах. Служит альтернативой методу помещения текстовых заметок непосредственно на диаграммах
5. Уточнение <i>Elaboration</i> <i>(ELAB)</i>	Для уточнения или более подробного описания изображаемого на диаграмме. Обычно используется для детального описания разветвления или слияния стрелок на перекрестках

Декомпозиция работ в IDEF3

- В IDEF3 декомпозиция используется для детализации работ.
- Методология IDEF3 позволяет декомпозировать работу **многократно**, т.е. работа может иметь множество дочерних работ.
- Это позволяет в одной модели описать **альтернативные потоки**.
- Возможность множественной декомпозиции предъявляет дополнительные требования к нумерации работ

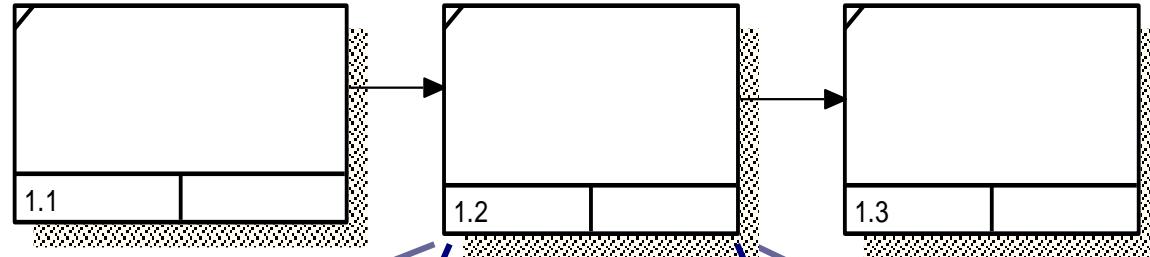
Нумерация работ в IDEF3

- Номер работы состоит из *номера родительской работы, версии декомпозиции и собственного номера работы на текущей диаграмме*

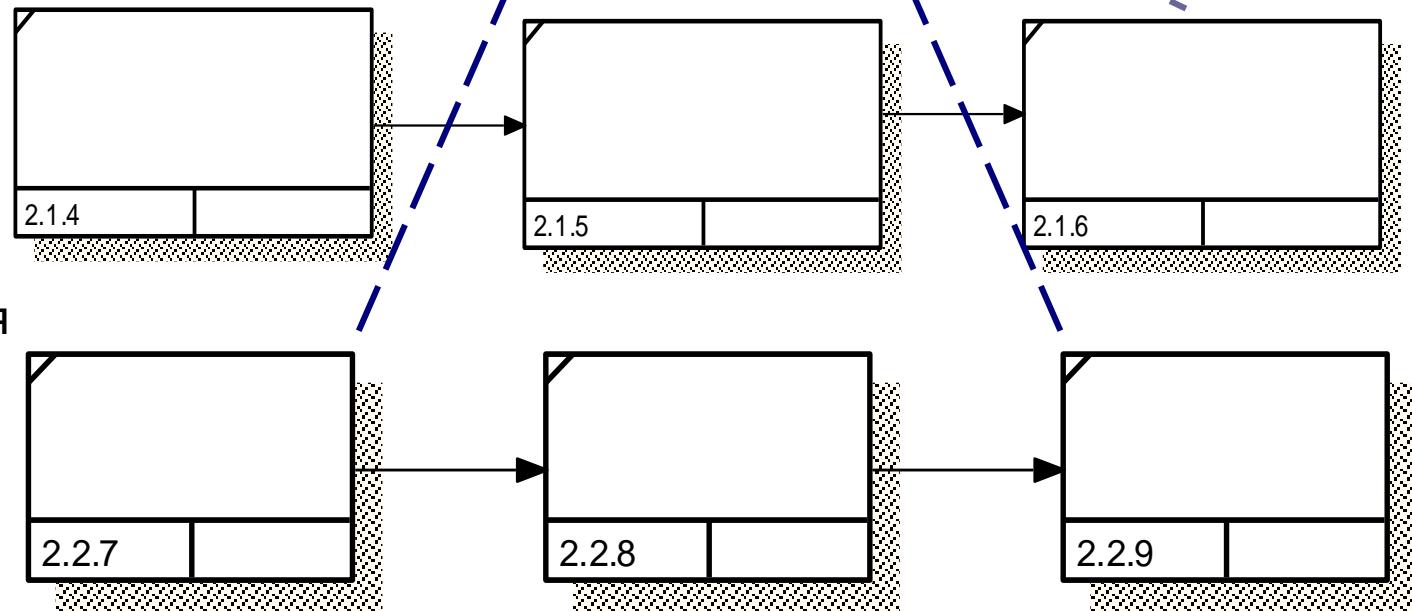


Структура множественной декомпозиции работ

Первая
декомпозиция
работы 1.2

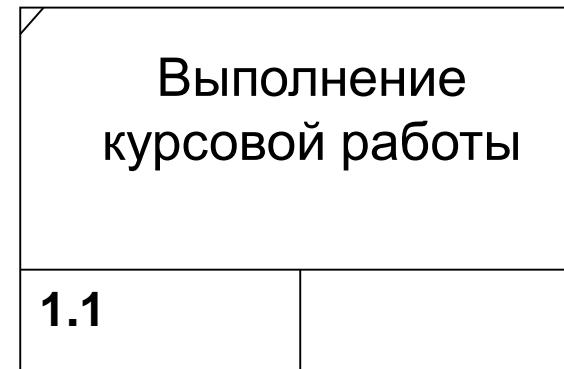


Вторая
декомпозиция
работы 1.2

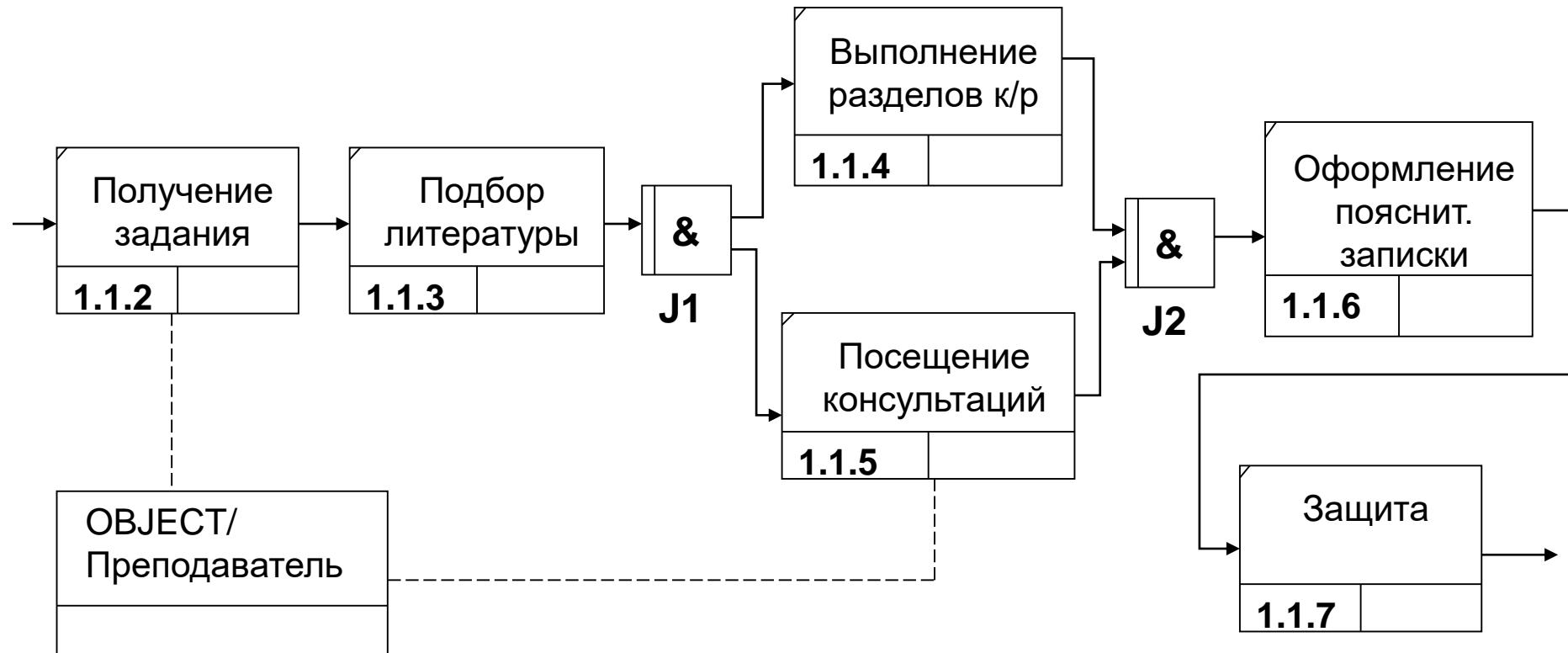


Пример построения модели IDEF3

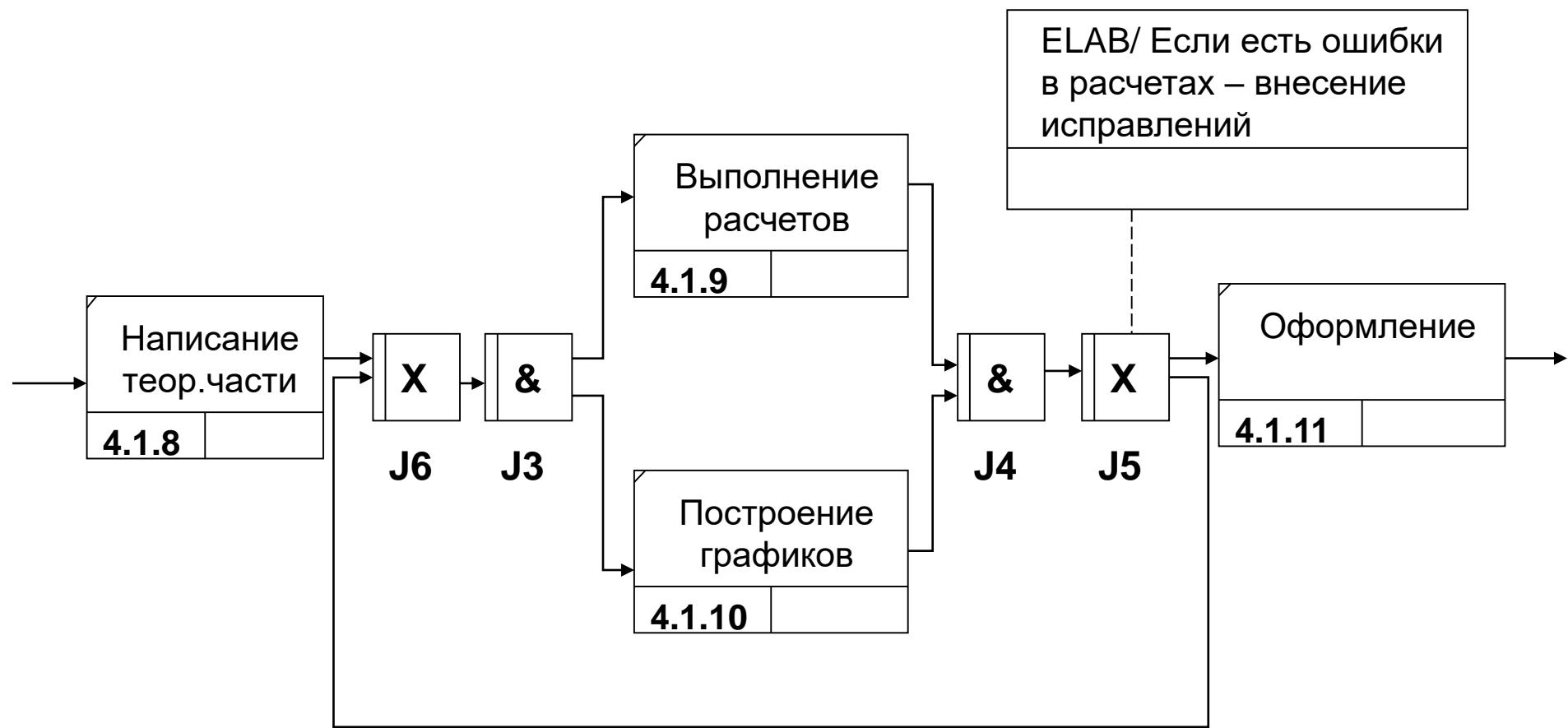
- Пример построения динамической модели процесса «Выполнение курсовой работы»
- Контекстная диаграмма



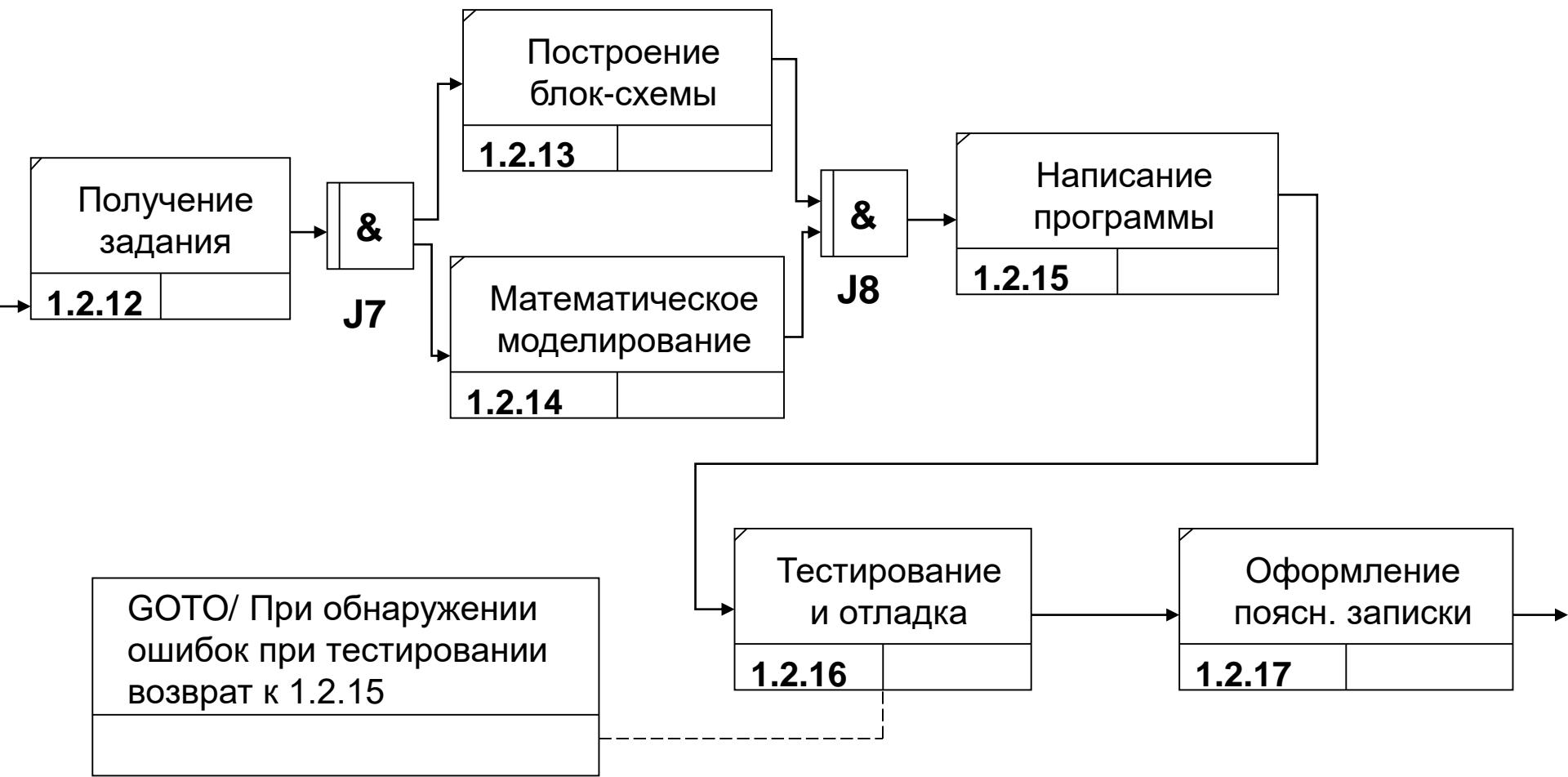
Пример построения модели IDEF3



Пример построения модели IDEF3



Пример построения модели IDEF3





Как объяснил
заказчик



Как понял
руководитель проекта



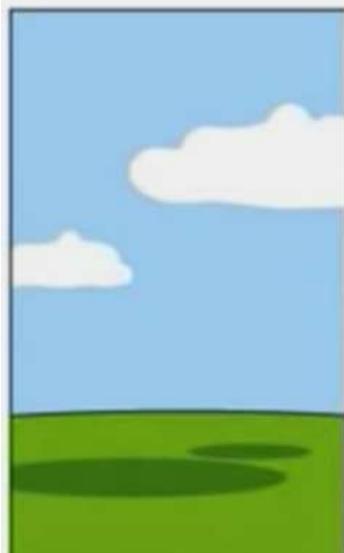
Как спроектировал
дизайнер



Как сделал
программист



Как описал
бизнес-консультант



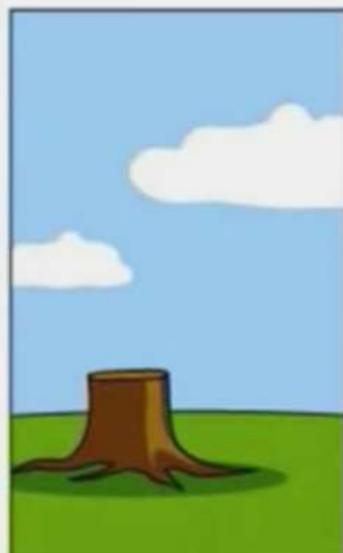
Как был
задокументирован



Что было сделано



За что заплатил
клиент



Какая была
поддержка



Что реально нужно
было заказчику

Но вы же утвердили
техническое задание !

Техническое задание ?
Мы думали, ТЗ - это «Точка зрения»
и у нас их уже несколько

5. ДОКУМЕНТИРОВАНИЕ ТРЕБОВАНИЙ

5. ДОКУМЕНТИРОВАНИЕ ТРЕБОВАНИЙ

Документы

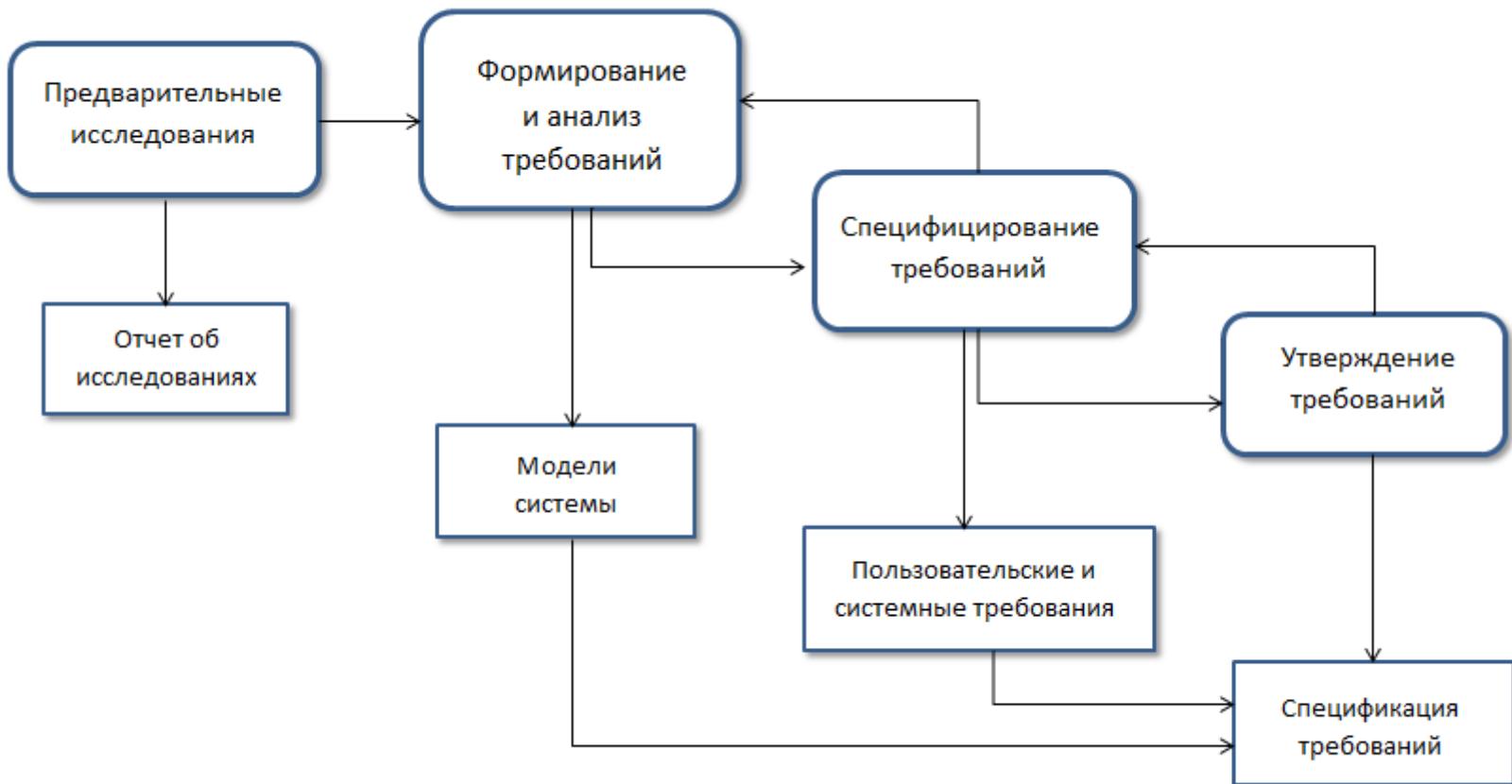
- **Обзор продукта.** Внешнее описание системы.
 - (1) документ, составленный на основании пожеланий заказчика, достаточно точно определяющий задачи разработчиков ПС
 - (2) постановка задачи, решение которой должно обеспечить разрабатываемое ПС
- Системные модели (диаграммы прецедентов, деятельности, user story)
- **Техническое задание.** Включает введение, основание для разработки, назначение разработки, требования к программе, требования к программной документации, технико-экономические показатели, стадии и этапы разработки, порядок контроля и приема
- Частные технические задания
- **Спецификация требований программного обеспечения** (англ. *Software Requirements Specification*, SRS) — законченное описание поведения программы, которую требуется разработать.

5. ДОКУМЕНТИРОВАНИЕ ТРЕБОВАНИЙ

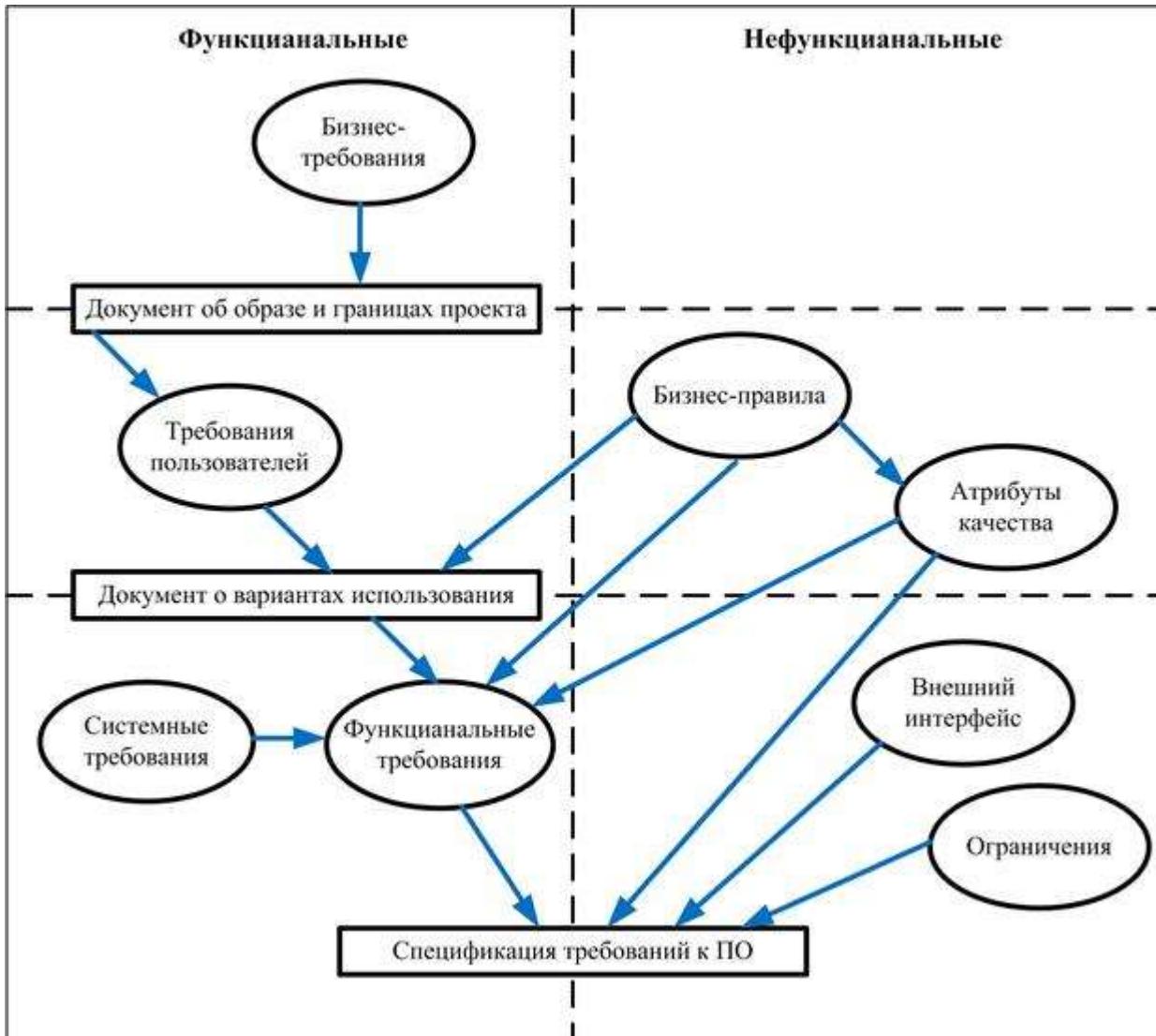
Стандарты

- ГОСТ 34.602-89 Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы.
- РД 50-34.698-90 Методические указания. Информационная технология. Комплекс стандартов и руководящих документов на автоматизированные системы. Автоматизированные системы. Требования к содержанию документов.
- Стандарт IEEE 830-1998. Методика составления спецификаций требований к программному обеспечению(заменен стандартом ISO/IEC/IEEE 29148:2011).

5. ДОКУМЕНТИРОВАНИЕ ТРЕБОВАНИЙ



5. ДОКУМЕНТИРОВАНИЕ ТРЕБОВАНИЙ



Преимущества

- Наличие четкого набора требований гарантирует, что команда разработчиков создаст программное обеспечение, отвечающее потребностям клиента.
- SRS поможет оценить стоимость работ и охватить объем проекта.
- Дает программистам представление о технологическом стеке, который им понадобится, и помогает планировать работу.
- Дизайнеры получают представление о проекте через документы SRS, чтобы они могли адаптировать дизайн к варианту использования.
- Тестировщики получают рекомендации по созданию тестовых случаев, соответствующих потребностям бизнеса.

1. Введение

Введение представляет собой обзор, помогающий читателям разобраться в структуре и принципе использования спецификации требований к ПО.

1.1 *Назначение*

Определите продукт или приложение, требования для которого указаны в этом документе, в том числе редакцию или номер выпуска. Если эта спецификация требований к ПО относится только к части системы, идентифицируйте эту часть или подсистему.

1.2 *Соглашения, принятые в документах*

Опишите все стандарты или типографические стандарты, включая стили текста, особенности выделения или замечания. Например, укажите, унаследован ли приоритет, указанный для требований высшего уровня, всеми их детализированными требованиями, или каждое положение о функциональных требованиях должно обладать собственным приоритетом.

1.3 *Предполагаемая аудитория и рекомендации по чтению*

Перечислите пользователей, для которых предназначена эта спецификация требований к ПО. Опишите содержание документа и его структуру.

Порекомендуйте наиболее подходящую для каждого класса читателей последовательность чтения документа.

1.4 Границы проекта

Кратко опишите ПО и его назначение. Покажите, как связан продукт с пользователями или корпоративными целями, а также с бизнес-целями и стратегиями. Если имеется отдельный документ об образе и границах проекта, не повторяйте его содержимое, а просто сошлитесь на него. Если спецификацию требований к ПО предполагается разрабатывать постепенно, она должна содержать собственное положение об образе и границах продукта в качестве подраздела долгосрочного стратегического образа.

1.5 Ссылки

Перечислите все документы или другие ресурсы, на которые вы ссылаетесь в этой спецификации, в том числе гиперссылки на них. Это могут быть руководства по стилям пользовательского интерфейса, контракты, стандарты, спецификации к системным требованиям, документы о вариантах использования, спецификации интерфейса, концептуальные документы и спецификация требований к ПО для продуктов, на которые вы ссылаетесь. Объем информации должен быть достаточным для того, чтобы пользователь сумел при необходимости получить доступ к каждому указанному материалу, а именно: название, имя автора, номер версии, дата и источник или расположение (например, сетевая папка или URL).

2. Общее описание

В этом разделе представлен общий обзор продукта и среды, в которой он будет применяться, предполагаемая пользовательская аудитория, а также известные ограничения, предположения и зависимости.

2.1 Общий взгляд на продукт

Опишите содержание и происхождение продукта. Поясните, является он новым членом растущего семейства продуктов, новой версией существующей системы, заменой существующего приложения или совершенно новым продуктом? Если спецификация требований определяет компонент более крупной системы, укажите, как это ПО соотносится со всей системой и определите основные интерфейсы между ними.

2.2 Особенности продукта

Перечислите основные особенности продукта или его главные функции. Детали будут изложены в разделе 3 спецификации требований к ПО, здесь же следует их только указать. Также здесь уместно проиллюстрировать основные группы требований и их взаимоотношения, например показать диаграмму потоков данных высшего уровня, диаграмму вариантов использования или диаграмму классов.

2.3 Классы и характеристики пользователей

Определите различные классы пользователей, которые, как предполагается, будут работать с вашим продуктом, и опишите их соответствующие характеристики. Некоторые требования могут относиться только к определенным классам пользователей. Определите привилегированные классы пользователей. Классы пользователей представляют подмножество заинтересованных в проекте лиц, их описание приводится в документе об образе и границах проекта.

2.4 Операционная среда

Опишите рабочую среду ПО, включая аппаратные средства, операционные системы и их версии, а также географическое местоположение пользователей, серверов и баз данных. Перечислите все остальные компоненты ПО или приложений, с которыми система должна быть совместима. В документе об образе и границах проекта эта информация может быть раскрыта более подробно.

2.5 Ограничения проектирования и реализации

Опишите любые факторы, которые ограничивают возможности, доступные разработчикам, и логически обоснуйте каждое положение. Ограничения могут быть такого рода:

- определенные технологии, средства, языки программирования и базы данных, которые следует использовать или избегать;
- ограничения, налагаемые операционной средой продукта, например типы и версии установленных Web-браузеров;
- обязательные соглашения или стандарты разработки (например, если обслуживать ПО будут клиенты, то они должны указать особенности дизайна и стандарты программирования, которые субподрядчик обязан соблюдать);
- обратная совместимость с продуктами, выпущенными ранее;
- ограничения, налагаемые бизнес-правилами (они должны быть зафиксированы в других документах, как рассказано в главе 9);
- ограничения, связанные с оборудованием, например требования к срокам, ограничения памяти или процессора, размер, вес, материалы или затраты;
- соглашения, связанные с пользовательским интерфейсом существующего продукта, которые необходимо соблюдать при улучшении существующего продукта;
- стандартный формат обмена данными, например XML.

2.6 Документация для пользователей

Перечислите все компоненты пользовательской документации, поставляемые с исполняемым ПО. В них могут входить руководства пользователя, онлайн-справка и обучающие программы. Определите все необходимые форматы, стандарты и средства поставки документации.

2.7 Предположения и зависимости

Предположением(assumption) называется положение, которое считается истинным при отсутствии доказательства или определяющей информации. Проблемы возможны в том случае, если предположение неверны, не находятся в совместном использовании или они изменяются, поэтому определенные предположения можно отнести к группе рисков проекта. Один пользователь спецификации может считать, что продукт будут соответствовать особому стандарту пользовательского интерфейса, тогда как другой предположит нечто совершенно иное. Разработчик может думать, что определенный набор функций написан специально для этого приложения, аналитик— что он будет взят из предыдущего проекта, а менеджер проекта— что предполагается приобрести коммерческую библиотеку функций.

3. Функции системы

Способы систематизации функциональных требований. По классификации: по вариантам использования, режиму работы, классам пользователей, стимулам, реакциям, классам объектов или функциональной иерархии (IEEE, 1998b). Возможны также комбинации этих элементов, например, варианты использования внутри классов пользователей. 3.x Функция системы X

Укажите название особенности несколькими словами, например «3.1 Проверка правописания». Так же назовите подразделы с 3.x.1 по 3.x.3 для каждой функции системы.

3.x.1 Описание и приоритеты

Кратко опишите особенность функции и укажите, обладает ли она высоким, средним или низким приоритетом. Приоритеты являются динамической характеристикой, они могут изменяться в ходе проекта. Если вы используете средство управления требованиями, определите атрибут требований для приоритета.

3.х.2 *Последовательности «воздействие - реакция»*

Перечислите последовательность воздействий, оказываемых на систему (действия пользователей, сигналы внешних устройств и др.), и отклики системы, определяющие реакцию конкретной функции. Эти воздействия соответствуют первоначальным шагам для вариантов использования или внешним системным событиям.

3.х.3 *Функциональные требования*

Перечислите по пунктам детализированные функциональные требования, которые связаны с этой особенностью. Здесь должны быть представлены определенные характеристики ПО, чтобы пользователь мог задействовать эту функцию или реализовать варианты использования. Опишите, как продукт должен реагировать на ожидаемые ошибки, неправильный ввод информации или неверные действия. Присвойте каждому функциональному требованию уникальное имя.

4. Требования к внешнему интерфейсу

«требования к внешнему интерфейсу определяют оборудование, ПО или элементы баз данных, с которыми система или компонент должны взаимодействовать...» Информация этого раздела позволяет вам быть уверенным, что система будет должным образом взаимодействовать с внешними компонентами. Если у разных частей продукта разные внешние интерфейсы, вставьте подобный раздел в детализированные требования для каждой такой части.

4.1 Интерфейсы пользователя

Опишите логические характеристики каждого пользовательского интерфейса, который необходим системе. Некоторые из них перечислены здесь:

- ссылки на стандарты графического интерфейса пользователей или стилевые рекомендации для семейства продукта, которые необходимо соблюдать;
- стандарты шрифтов, значков, названий кнопок, изображений, цветовых схем, последовательностей полей вкладок, часто используемых элементов управления и т.п.; конфигурация экрана или ограничения разрешения;
- стандартные кнопки, функции или ссылки перемещения, одинаковые для всех экранов, например кнопка справки;
- быстрые клавиши;
- стандарты отображения сообщений;
- стандарты конфигурации для упрощения локализации ПО;
- специальные возможности для пользователей с проблемами со зрением.

4.2 *Интерфейсы оборудования*

Опишите характеристики каждого интерфейса между компонентами ПО и оборудования системы. В описание могут входить типы поддерживаемых устройств, взаимодействия данных и элементов управления между ПО и оборудованием, а также протоколы взаимодействия, которые будут использоваться.

4.3 *Интерфейсы ПО*

Опишите соединения продукта и других компонентов ПО (идентифицированные по имени и версии), в том числе базы данных, операционные системы, средства, библиотеки и интегрированные коммерческие компоненты. Укажите назначение элементов сообщений, данных и элементов управления, обмен которыми происходит между компонентами ПО. Опишите службы, необходимые внешним компонентам ПО, и природу взаимодействия между компонентами. Определите данные, к которым будут иметь доступ компоненты ПО. Если механизм предоставления общего доступа к данным должен быть реализован определенным способом, например в качестве глобальной области данных, то укажите его как ограничение.

4.4 *Интерфейсы передачи информации*

Укажите требования для любых функций взаимодействия, которые будут использоваться продуктом, включая электронную почту, Web-браузер, протоколы сетевого соединения и электронные формы. Определите соответствующие форматы сообщений. Опишите особенности безопасности взаимодействия или шифрования, частоты передачи данных и механизмов синхронизации.

5. Другие нефункциональные требования

5.1 Требования к производительности

Укажите специальные требования к производительности для различных системных операций. Обоснуйте их необходимость для того, чтобы помочь разработчикам принять правильные решения, касающиеся проектирования. Например, из-за жестких требований к времени отклика базы данных разработчики могут зеркализовать базу данных в нескольких географических метаположениях или денормализовать связанные таблицы баз данных для получения более быстрого ответа на запрос вашего проекта. Пропустите этот раздел, если все необходимые требования уже расписаны в других разделах.

5. СПЕЦИФИКАЦИЯ ТРЕБОВАНИЙ (ТЕХНИЧЕСКОЕ ЗАДАНИЕ)

Бизнес-правила

- **Бизнес-правила** — политика, руководящие принципы или положения, которые определяют или ограничивают некоторые аспекты бизнеса, в т.ч. правила, определяющие состав и правила выполнения определенных бизнес-процессов. К бизнес-правилам относятся корпоративные политики, правительственные постановления, промышленные стандарты и вычислительные алгоритмы, которые используются при разработке продукта или системы либо непосредственно влияют на разработку.
- Примеры бизнес-правил: «При отгрузке заказа менеджер должен запросить у бухгалтера товарно-транспортную накладную и счет-фактуру», «Если оплата по счету не поступила в течение 15 дней, заказ считается отменённым»

5. СПЕЦИФИКАЦИЯ ТРЕБОВАНИЙ (ТЕХНИЧЕСКОЕ ЗАДАНИЕ)

Пример бизнес-правил

Сайт интернет-магазина должен соответствовать требованиям федеральных законов:

- №152-ФЗ «О персональных данных» при хранении и обработке персональных данных покупателей;
- №54-ФЗ «О применении контрольно-кассовой техники...» при обработке платежей, выполняемых с использованием электронных денег и банковских карт.

5. СПЕЦИФИКАЦИЯ ТРЕБОВАНИЙ (ТЕХНИЧЕСКОЕ ЗАДАНИЕ)

- **Ограничения** — условия, ограничивающие выбор возможных решений по реализации отдельных требований или их наборов. Они существенно ограничивают выбор средств, инструментов и стратегий при разработке внешнего вида и структуры (в т.ч. архитектуры) продукта или системы.
- Примеры ограничений: «При аутентификации пользователя должны использоваться биометрические методы идентификации».

- **Внешние интерфейсы** — описание аспектов взаимодействия с другими системами и операционной средой. К ним относятся требования к API продукта или системы, а также требования к API других систем, с которыми осуществляется интеграция.
- Примеры внешних интерфейсов: «Обеспечить запись в журнал операционной системы следующих событий: сообщения о запуске и остановке сервиса XX»; «Обеспечить запись в журнал параметров модулей программы: сканера, ядра, антивирусных баз (информация должна заноситься в журнал при запуске программы и при обновлении модулей)»

Правила, ограничения

- Анкета должна содержать файл с фото, так как фото необходимо при оформлении документов.
- У пользователя должна быть возможность прикрепить фото к анкете.
- ПС должно иметь функционал прикрепления фото к анкете
- Фото должны храниться все фото в отдельной таблице в БД.
- Размер загружаемого фото: не более 10Мб.

Роль: ученик

Роли и права доступа

Функция	Авторизованный пользователь	Неавторизованный пользователь
Авторизация в системе	-	+
Восстановление пароля	-	+
Выход из системы	+	-
Регистрация	-	+

Пример пользовательских требований

Пошаговый сценарий покупки авиабилета на выбранный маршрут.

Набор user stories для покупки на сайте интернет-магазина:

- для сравнения товаров по характеристикам;
- для добавления товара в корзину;
- для управления корзиной;
- для оформления заказа;
- для онлайн-оплаты.

Авторизация через форму на сайте

Пользовательская история	Я, как неавторизованный, но зарегистрированный пользователь, должен иметь возможность войти в личный кабинет, чтобы начать использовать сервис.												
Бизнес-правила	<ul style="list-style-type: none">• ученик заходит на сайт → нажимает кнопку "Войти" → появляется окно входа• ученик вводит логин и пароль<ul style="list-style-type: none">◦ если ученик забыл пароль, то он нажимает кнопку "Восстановить пароль":<ul style="list-style-type: none">■ система отправляет на email ученика письмо с ссылкой для восстановления пароля■ ученик переходит по ссылке из письма → задает новый пароль → нажимает кнопку "Сохранить" → переходит на главную страницу сайта → нажимает кнопку "Войти" → появляется окно входа■ ученик вводит логин и пароль• Система проверяет зарегистрирован ли пользователь с указанными данными:<ul style="list-style-type: none">◦ зарегистрирован → ученик авторизуется в системе и переходит в личный кабинет◦ незарегистрирован → ученик переходит к форме регистрации												
Валидация	<table border="1"><thead><tr><th>Поле</th><th>Условие</th><th>Сообщение об ошибке</th></tr></thead><tbody><tr><td>Все поля</td><td>Обязательные</td><td>Пожалуйста, заполните все поля, чтобы войти в личный кабинет</td></tr><tr><td>Email</td><td>Пользователь существует в системе</td><td>Неверный email или пароль</td></tr><tr><td></td><td>email соответствует формату:</td><td>Неверный формат email</td></tr></tbody></table>	Поле	Условие	Сообщение об ошибке	Все поля	Обязательные	Пожалуйста, заполните все поля, чтобы войти в личный кабинет	Email	Пользователь существует в системе	Неверный email или пароль		email соответствует формату:	Неверный формат email
Поле	Условие	Сообщение об ошибке											
Все поля	Обязательные	Пожалуйста, заполните все поля, чтобы войти в личный кабинет											
Email	Пользователь существует в системе	Неверный email или пароль											
	email соответствует формату:	Неверный формат email											

Примеры ограничений

Сервер приложений сайта должен разрабатываться на языке Java.

Сайт должен устанавливаться на ОС Ubuntu Server 14.04.



Этап проектирования ПО

Вопросы:

1. Особенности процесса проектирования ПО
2. Модульное проектирование программных средств
3. Оценка корректности и эффективности структурного разбиения программы на модули

1. ОСОБЕННОСТИ ПРОЦЕССА ПРОЕКТИРОВАНИЯ ПО

Проектирование объясняет, как ПС будет удовлетворять предъявленным к нему требованиям.

Проектирование — итерационный процесс, при помощи которого требования к ПС транслируются в инженерные представления ПС.

Процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или ее компонентов называется **проектированием**.

Программная инженерия. Проектирование программного обеспечения (SWEBOk, Software Design)

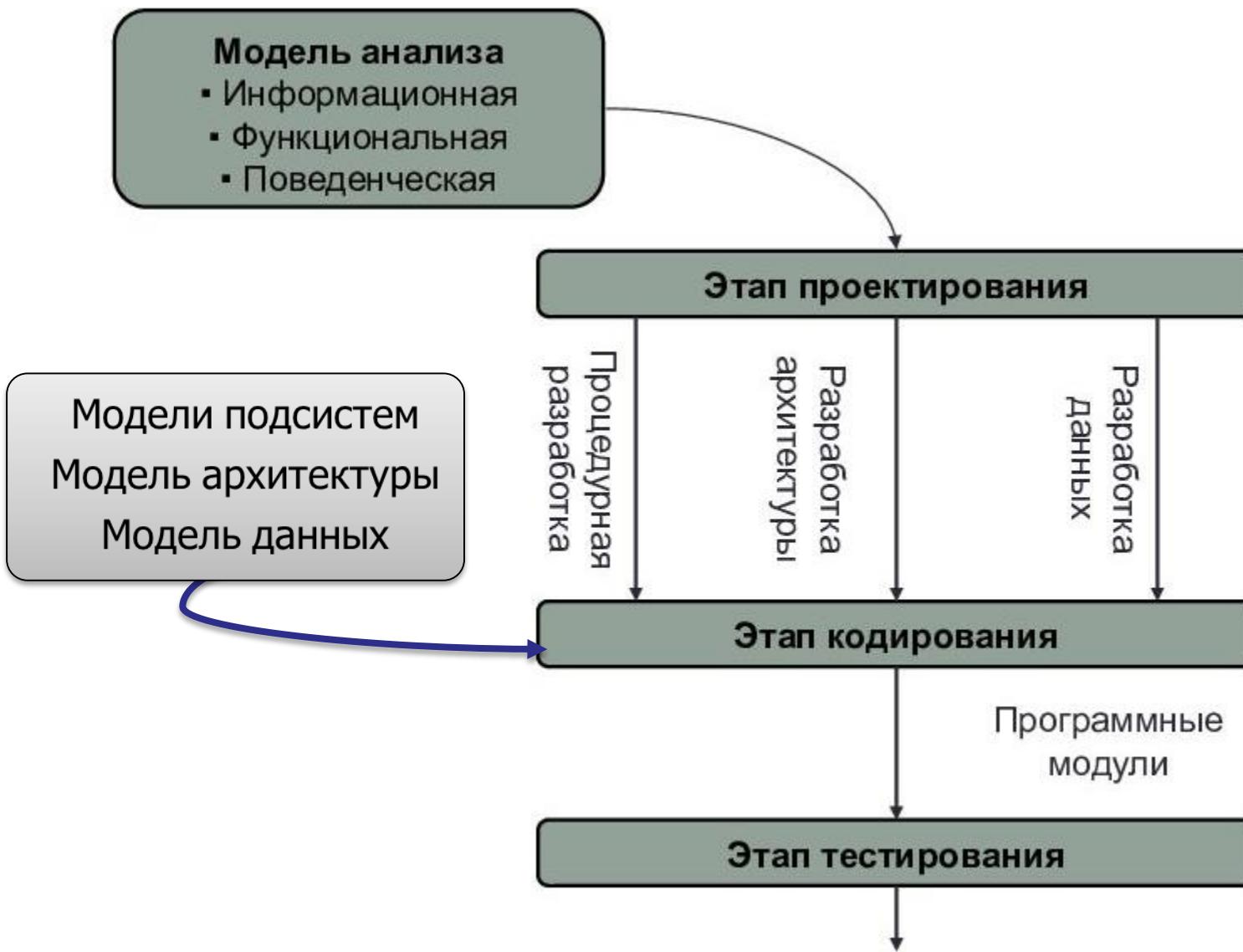
1. ОСОБЕННОСТИ ПРОЦЕССА ПРОЕКТИРОВАНИЯ ПО

Что делаем (описание продукта, функционала, пользователей)?

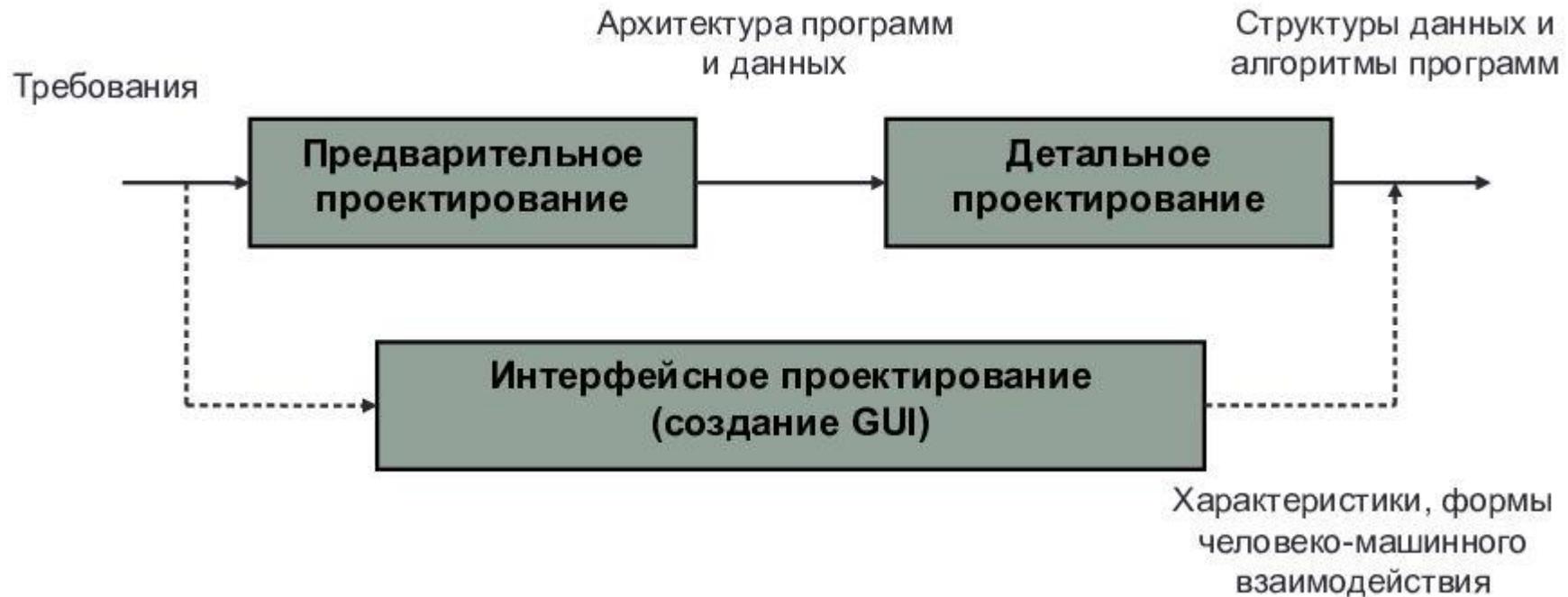
Как делаем (архитектура)?

Как проверить, что цель достигнута (тестирование, критерии оценки)?

Особенности процесса проектирования ПО



Особенности процесса проектирования ПО



- *Структурирование системы*
- *Моделирование управления*
- *Декомпозиция подсистем на модули*

1. ОСОБЕННОСТИ ПРОЦЕССА ПРОЕКТИРОВАНИЯ ПО

Проектирование состоит из двух составных частей:

- **Архитектурный** или высокоуровневый дизайн (software architectural design, top-level design) – описание высокоуровневой структуры и организации компонентов системы;
- **Детализированная архитектура** (software detailed design) – описывающая каждый компонент в том объеме, который необходим для конструирования.

1. ОСОБЕННОСТИ ПРОЦЕССА ПРОЕКТИРОВАНИЯ ПО

Архитектура (лат. *architectura*, хотя происходит от греческих корней *архι* и *тεκτονική*) – высшее плотничное или строительное искусство.

Архитектура — это описание (модель) основной компоновки и взаимодействия частей системы (будь то физический либо абстрактный объект или сущность).

ISO 15704

Архитектура – это базовая организация системы, воплощенная в ее компонентах, их отношениях между собой и с окружением, а также принципы, определяющие проектирование и развитие системы.

Стандарт IEEE 1472000 и IEEE 1471 “Рекомендуемые методы описания архитектуры преимущественно-программных систем”

1. ОСОБЕННОСТИ ПРОЦЕССА ПРОЕКТИРОВАНИЯ ПО

Под **архитектурой** понимается набор основных правил, определяющих организацию системы:

- совокупность структурных элементов системы и связей между ними;
- поведение элементов системы в процессе их взаимодействия;
- иерархию подсистем, объединяющих структурные элементы;
- архитектурный стиль (используемые методы и средства описания архитектуры, а также архитектурные образцы)

1. ОСОБЕННОСТИ ПРОЦЕССА ПРОЕКТИРОВАНИЯ ПО

Рамочная модель разработки архитектуры



1. ОСОБЕННОСТИ ПРОЦЕССА ПРОЕКТИРОВАНИЯ ПО

Архитектурное представление — это упрощенное описание (абстракция) системы с конкретной точки зрения:

- поведенческой (динамической),
- структурной (статической),
- логической (удовлетворение функциональным требованиям),
- физической (распределенность),
- реализации (как детали архитектуры представляются в коде) и т.п.,

охватывающее определенный круг интересов и опускающее объекты, несущественные с данной точки зрения.

1. ПРИНЦИПЫ ПРОЕКТИРОВАНИЯ

1. **Абстракция**, как результат процесса абстракции – модель, упрощающая поставленную проблему до рамок, значимых для заданного контекста.
2. **Декомпозиция и разбиение на модули** (Decomposition and Modularization).
3. **Сцепление** (Coupling) – определяет силу связи (часто, взаимного влияния) между модулями. **Связность** (Cohesion) определяет как тот или иной элемент обеспечивает связь внутри модуля, внутреннюю связь.
4. **Инкапсуляция/сокрытие информации** (Encapsulation/information hiding)
5. **Разделение интерфейса и реализации** (Separation of interface and implementation). Данная техника предполагает определение компонента через специфицирование интерфейса, известного (описанного) и доступного клиентам (или другим компонентам), от непосредственных деталей реализации.

1. Хорошая архитектура:

Масштабируемость (Scalability)

возможность расширять систему и увеличивать ее производительность, за счет добавления новых модулей.

Ремонтопригодность (Maintainability)

изменение одного модуля не требует изменения других модулей

Заменимость модулей (Swappability)

модуль легко заменить на другой

Возможность тестирования (Unit Testing)

модуль можно отсоединить от всех остальных и протестировать / починить

Переиспользование (Reusability)

модуль может быть переиспользован в других программах и другом окружении

Сопровождаемость (Maintenance)

разбитую на модули программу легче понимать и сопровождать

1. Свойства хорошо спроектированного модуля:

Функциональная целостность и завершенность — каждый модуль реализует одну функцию, но реализует хорошо и полностью; модуль самостоятельно (без помощи дополнительных средств) выполняет полный набор операций для реализации своей функции

Один вход и один выход — на входе программный модуль получает определенный набор исходных данных, выполняет содержательную обработку и возвращает один набор результатных данных, т.е. реализуется стандартный принцип — *вход–процесс–выход*;

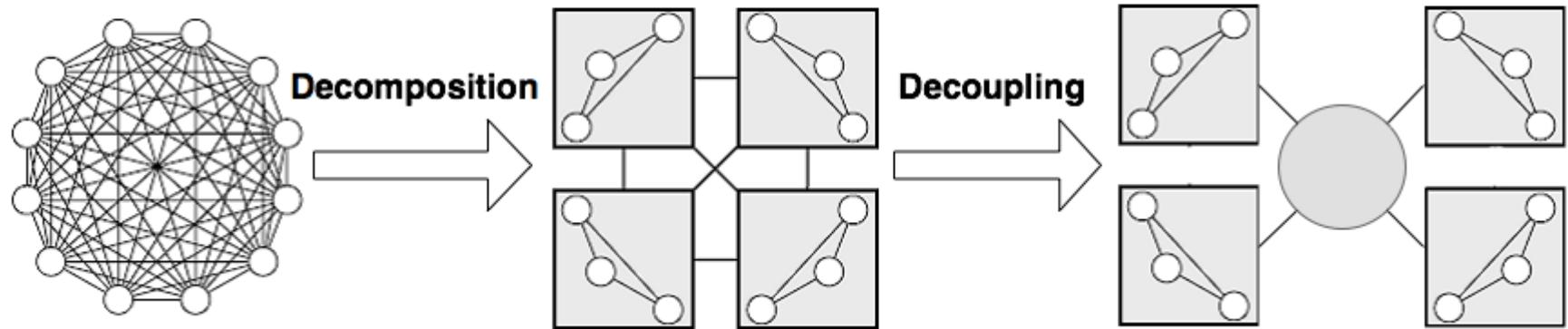
Логическая независимость — результат работы программного модуля зависит только от исходных данных, но не зависит от работы других модулей

Слабые информационные связи с другими модулями — обмен информацией между модулями должен быть по возможности минимизирован.

2. Модульная структура программных средств

Декомпозиция - основа модульной архитектуры

Создание Архитектуры

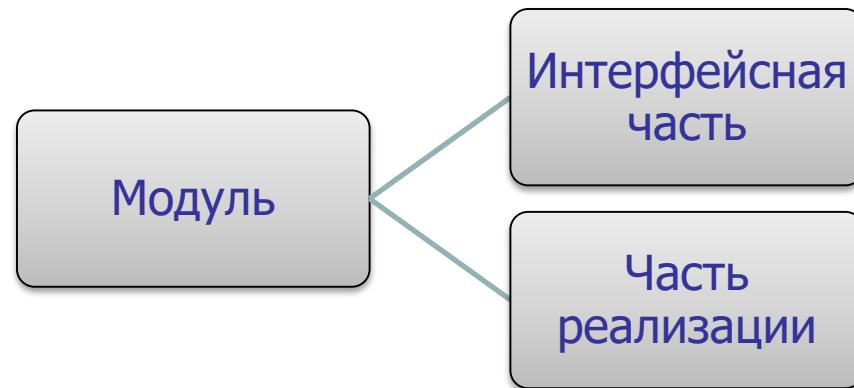


2. Модульная структура программных средств

Каждый модуль должен отвечать за решение какой-то подзадачи и выполнять соответствующую ей функцию.

Помимо функционального назначения модуль характеризуется также набором данных, необходимых ему для выполнения его функции, то есть:

Модуль = Функция + Данные, необходимые для ее выполнения.



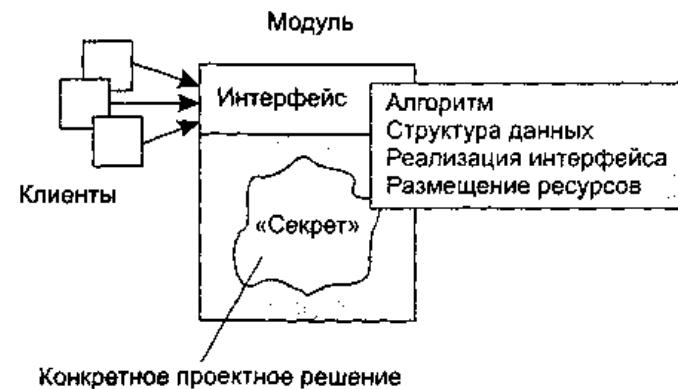
2. Модульное проектирование программных средств

Принцип информационной закрытости (автор — Д. Парнас, 1972) :
содержание модулей должно быть скрыто друг от друга.

Модуль должен определяться и проектироваться так, чтобы его содержимое (процедуры и данные) было недоступно тем модулям, которые не нуждаются в такой информации (клиентам).

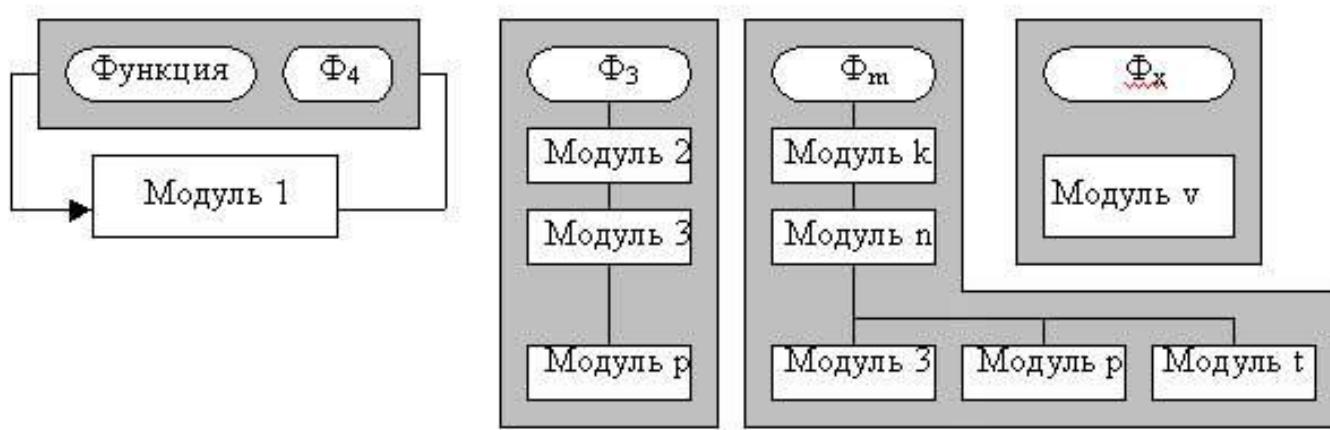
Информационная закрытость означает :

- 1) все модули независимы, обмениваются только информацией, необходимой для работы;
- 2) доступ к операциям и структурам данных модуля ограничен.



2. Преобразование функций в модули

Отдельный элемент функционально – структурного анализа описывает *законченную содержательную функцию обработки информации*, которая предполагает определенный способ реализации на программном уровне.



Функции ввода-вывода информации рекомендуется отделять от функций вычислительной или логической обработки данных.

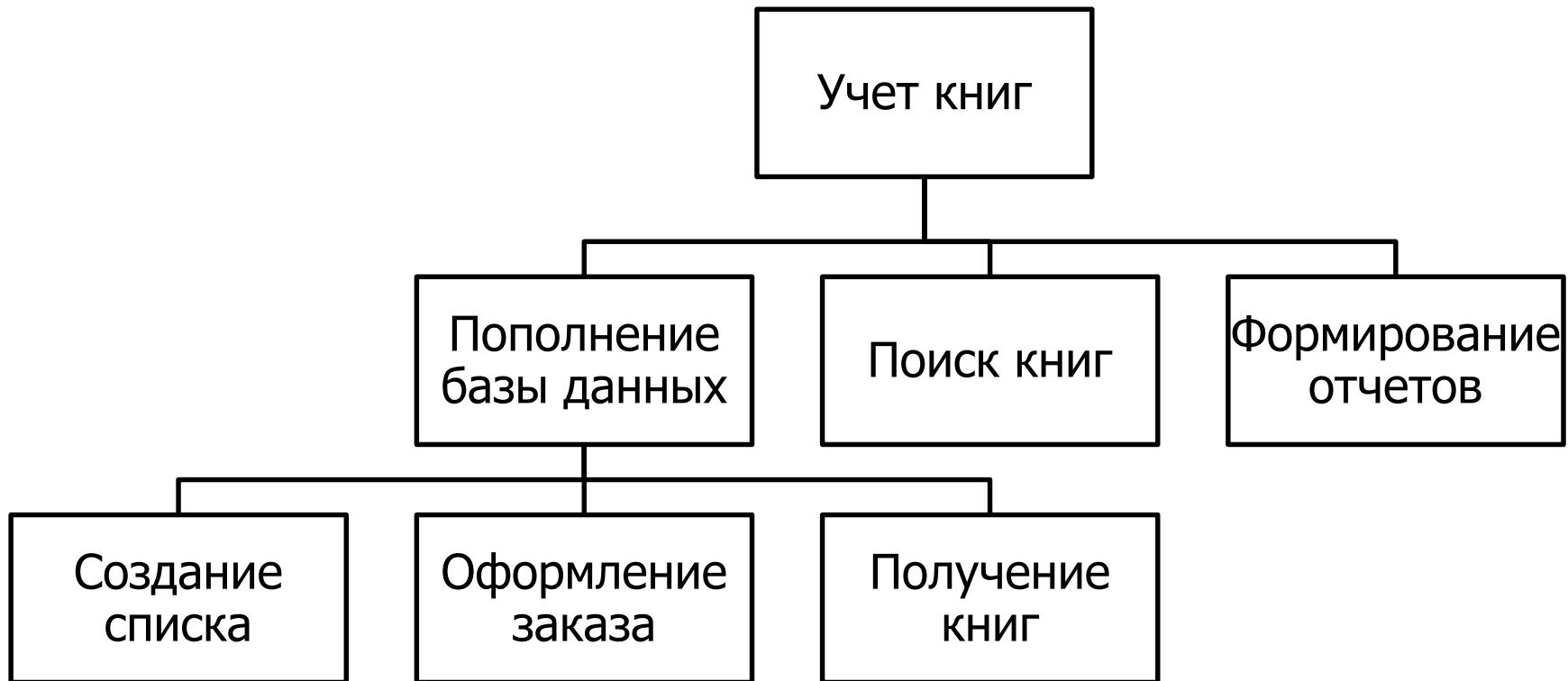
2. Преобразование функций в модули

Иерархия функций



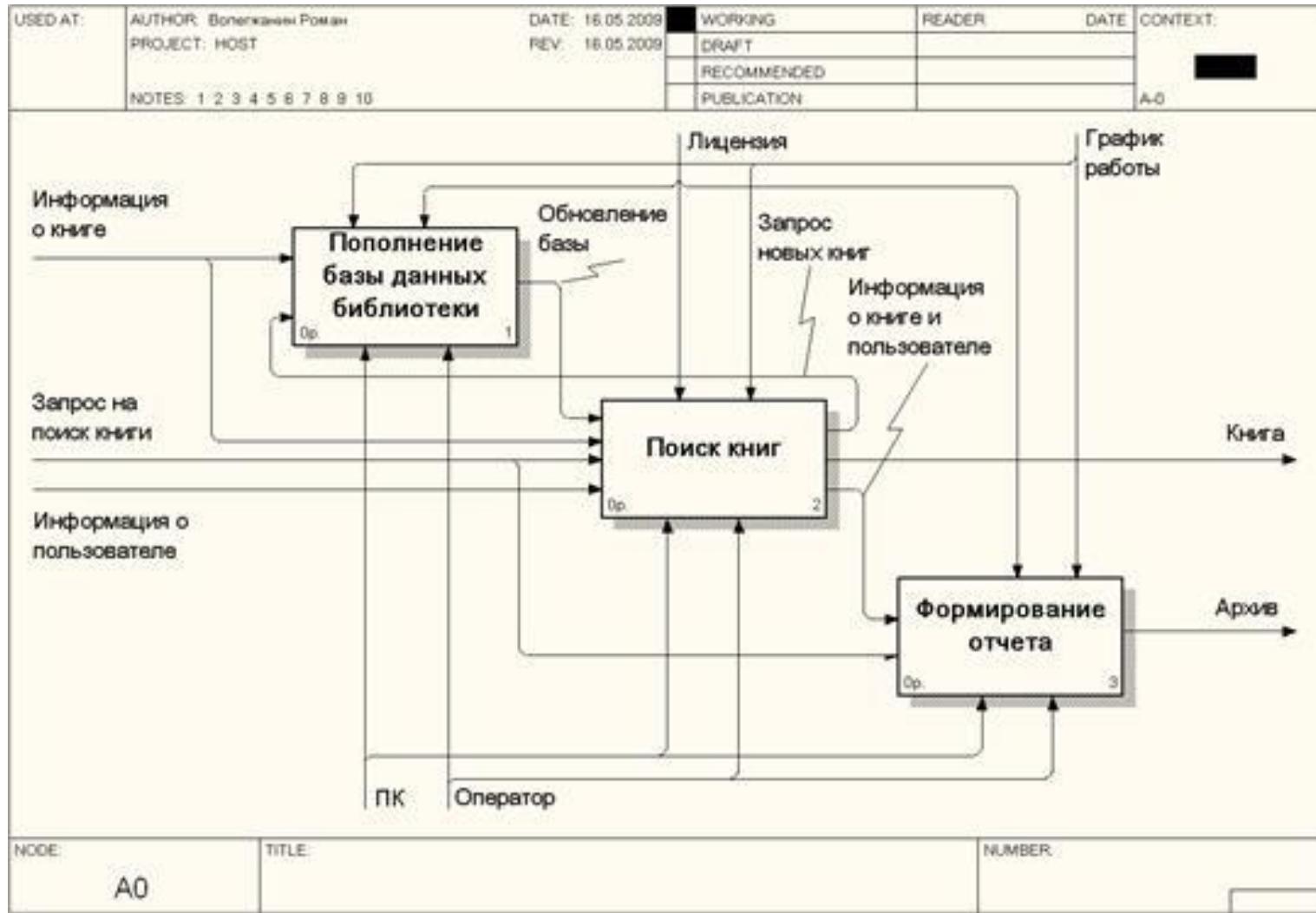
При отображении функций в модули необходимо получить схему, которая ставит в соответствие каждой функции определенный модуль.

Пример



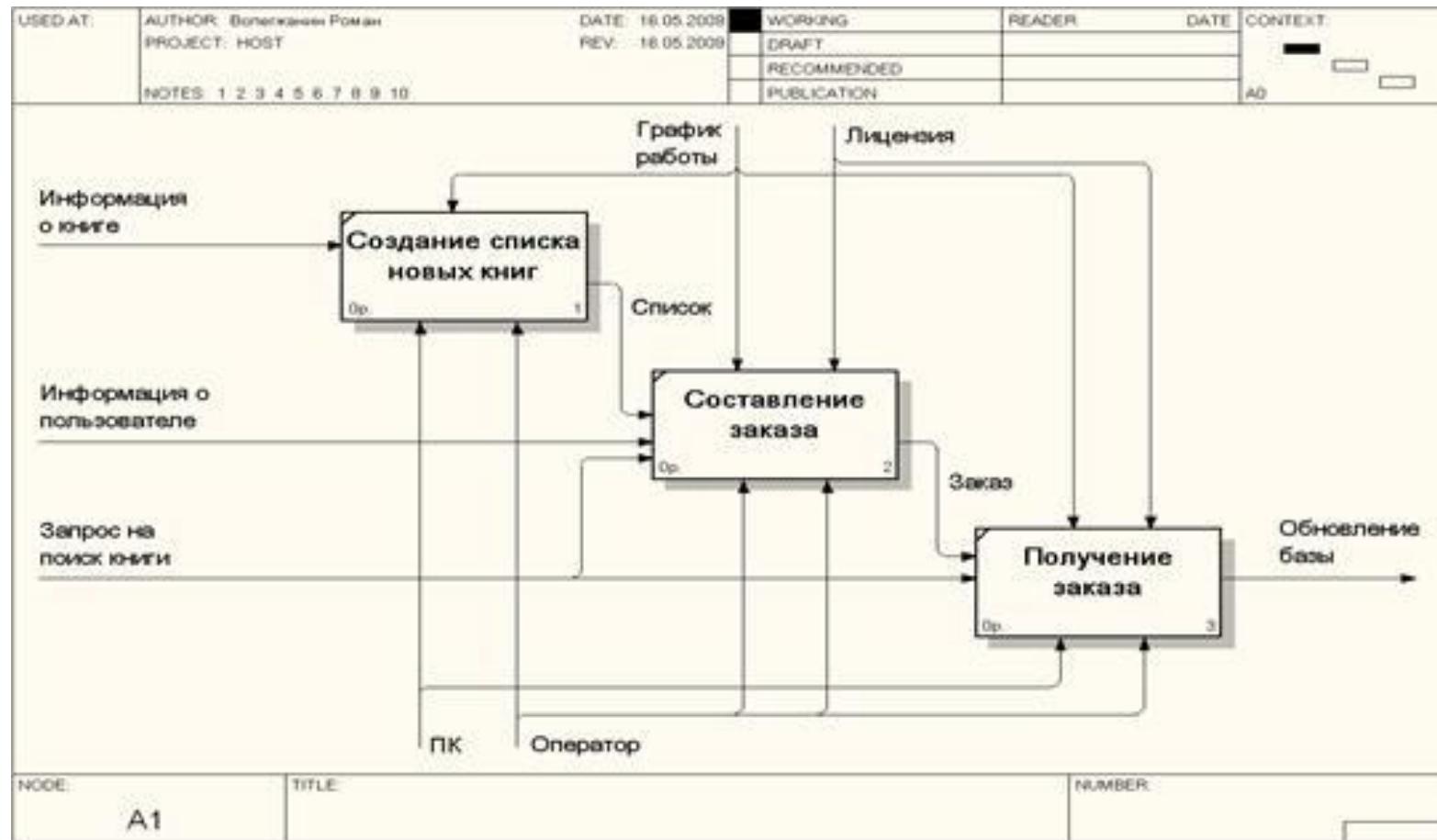
Преобразование функций в модули

Функция - данные



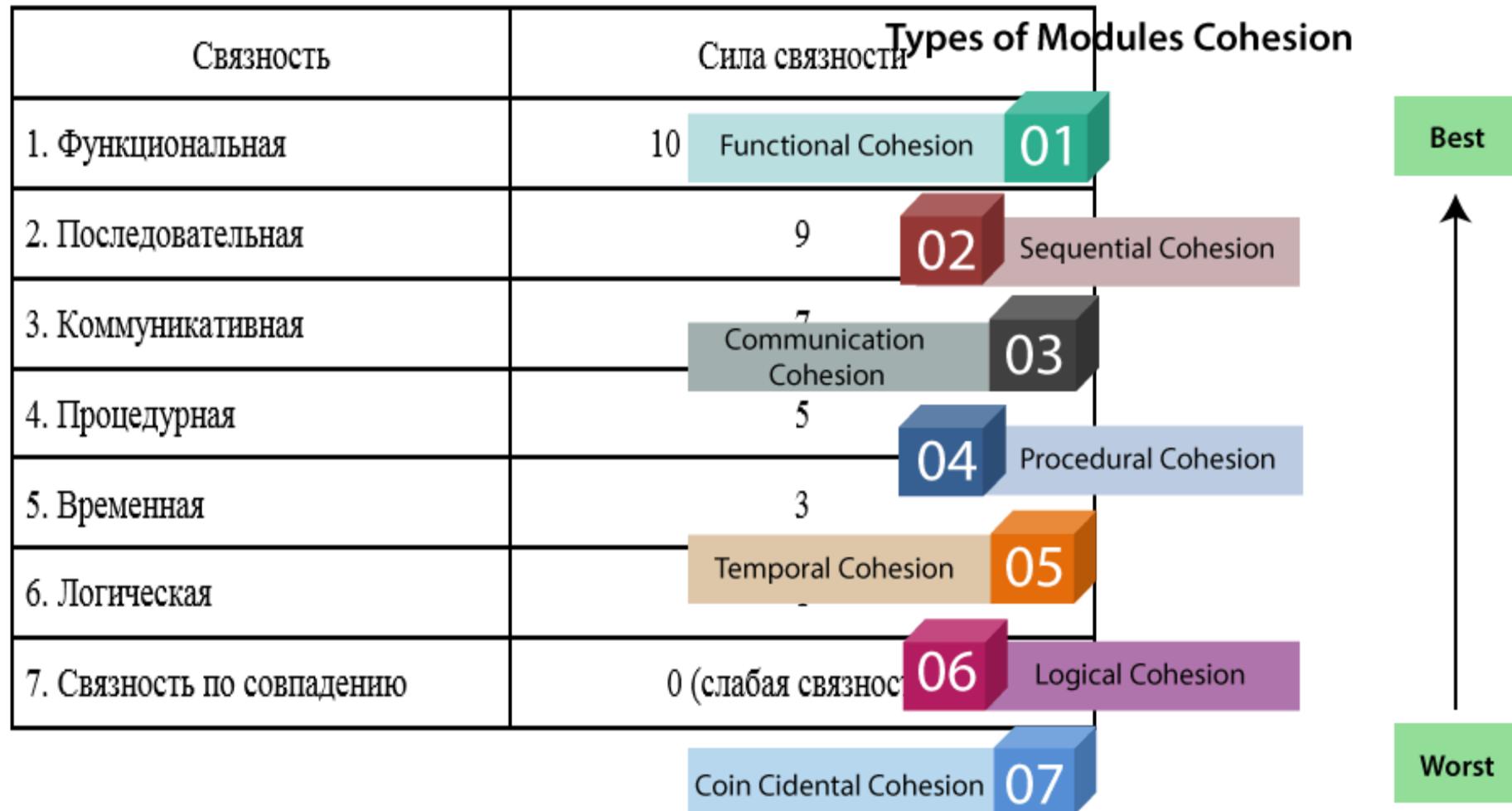
Преобразование функций в модули

Функция - данные



3. Связность модуля

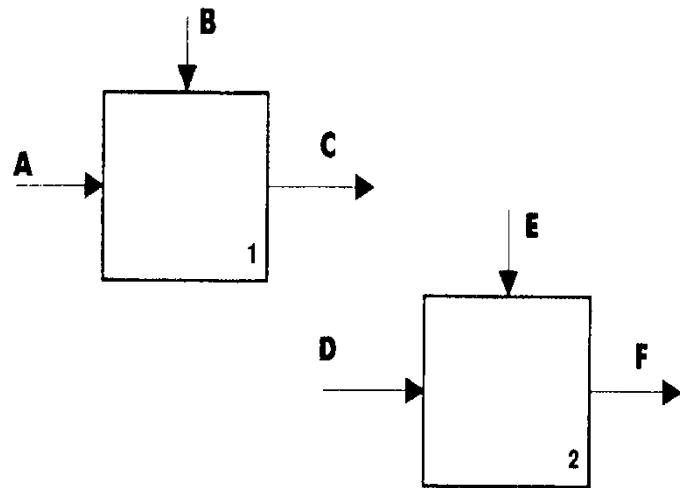
Типы и силы связности модулей



3. Связность модуля

Связность по совпадению ($CC=0$). В модуле отсутствуют явно выраженные внутренние связи.

Логическая связность ($CC=1$). Части модуля объединены по принципу функционального подобия.



3. Связность модуля

Элементы **логически-связного** модуля принадлежат к одной категории, но клиент должен сам выбрать выполняемое действие в зависимости от контекста.

Фактически, модуль представляет собой «белый ящик».

Это приводит к плохому интерфейсу модуля, с различными параметрами и методами для выполнения одного и того-же действия

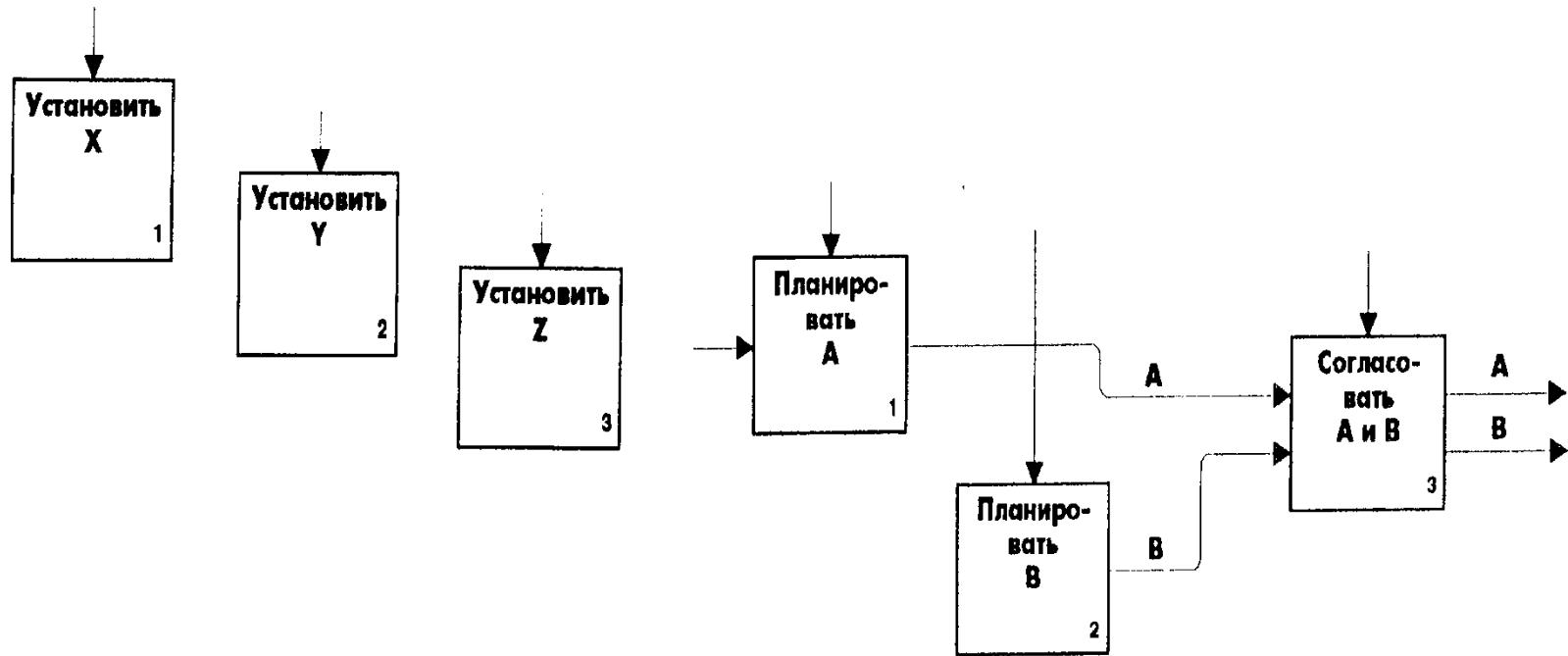
Модуль **Пересылка сообщения**
переслать по E-mail
переслать по факсу
послать в телеконференцию
послать по FTP

Конец модуля

3. Связность модуля

Временная связность (CC=3). Части модуля не связаны, но необходимы в один и тот же момент времени.

Процедурная связность (CC=5). Части модуля связаны порядком выполняемых действий, реализующих сценарий поведения.



3. Связность модуля

Временная связность

Элементы такого модуля должны выполняться в примерно в одно время.

Модуль представляет собой «белый ящик» или «серый ящик» в зависимости от качества реализации методов.

Модуль **Инициализировать систему**

Инициализация жесткого диска

Инициализация сетевого интерфейса

Включить индикатор Num Lock

Инициализация гибких дисков

Проверить системное время

Конец модуля

3. Связность модуля

Процедурная связность

Пограничное качество проектирования, может повлечь за собой плохую сопровождаемость кода.

Модуль состоит из элементов, реализующих независимые действия, но для которых важен порядок передачи управления.

При реализации может возникнуть дублирование кода, если 2 модуля работают с разными данными.

Модуль А

```
ОбработкаДанныхА  
СчитатьДанные  
ОбработатьДанныеA  
СохранитьДанные
```

Конец модуля

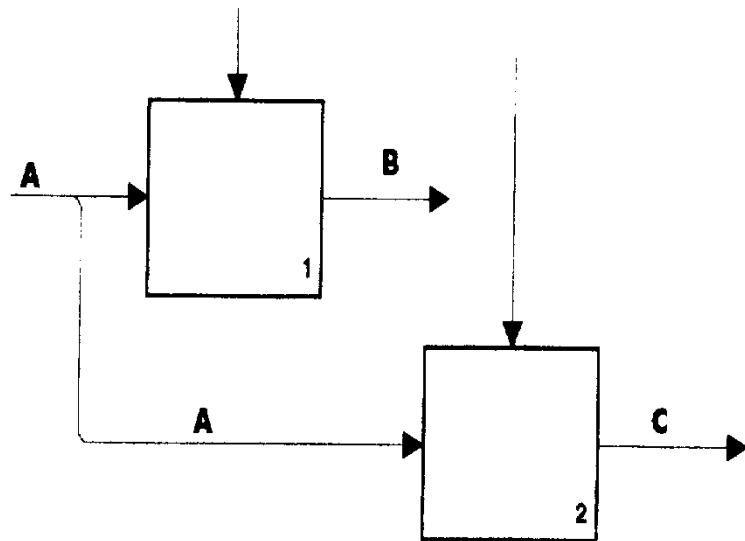
Модуль Б

```
ОбработкаДанныхБ  
СчитатьДанные  
ОбработатьДанныеB  
СохранитьДанные
```

Конец модуля

3. Связность модуля

Коммуникативная связность ($CC=7$). Части модуля связаны по данным (работают с одной и той же структурой данных).



3. Связность модуля

Элементы-обработчики используют одни и те же (может быть внешние) данные или участвуют в формировании общей структуры данных.

При использовании, клиенту может быть предоставлено избыточное количество данных.

Почти всегда разделение коммуникативно-связанного модуля на функционально-связанные приводит к улучшению сопровождаемости.

Модуль Успеваемость студентов

Сгенерировать отчет по успеваемости

Выдать отстающих студентов

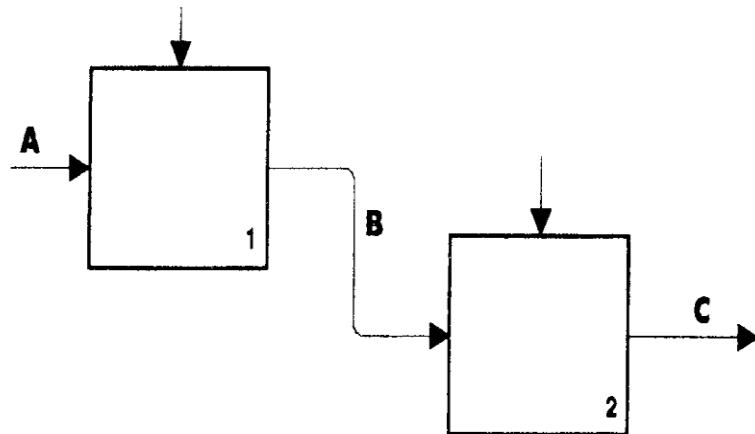
Выдать список отличников

Конец модуля

3. Связность модуля

Информационная (последовательная) связность ($CC=9$).

Выходные данные одной части модуля используются как входные данные другой части модуля.



3. Связность модуля

При последовательной связности элементы-обработчики образуют конвейер для обработки данных – результаты одного являются исходными данными для другого.

Хороший уровень связности. Но возможности повторного использования кода ограничены.

Модуль **Прием и проверка записи**

Прочитать запись из файла

Проверить контрольные данные

Удалить контрольные поля

Вернуть обработанную запись

Конец модуля

3. Связность модуля

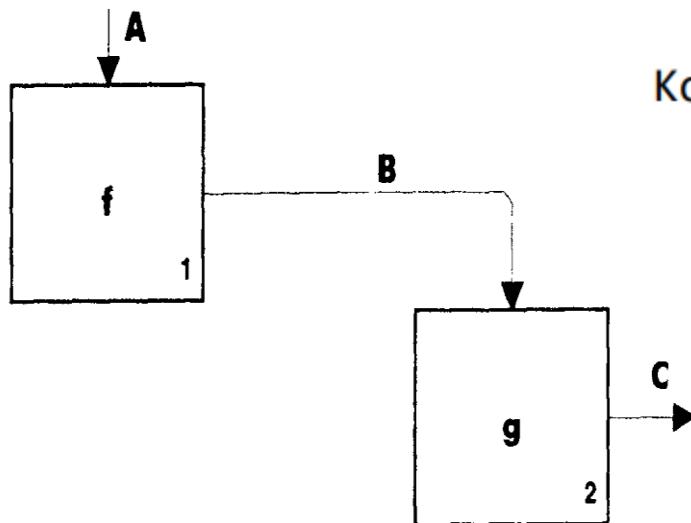
Функциональная связность (СС= 10). Части модуля вместе реализуют одну функцию.

Модуль Шифрование данных

- + Зашифровать данные
- + Расшифровать данные
- Сформировать набор ключей

...

Конец модуля



3. Определения связности модуля

Алгоритм определения уровня связности модуля:

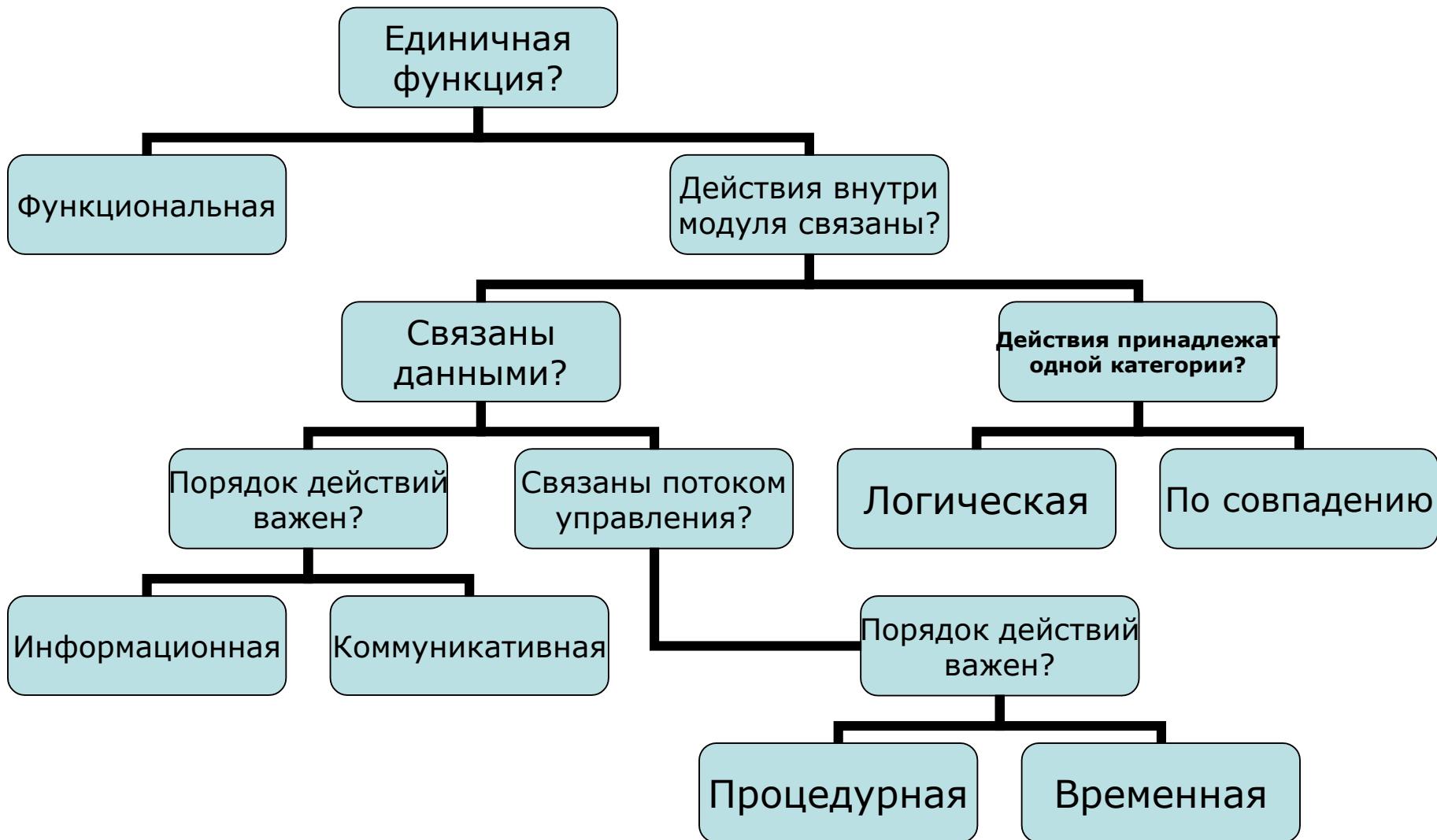
1. Если модуль — единичная проблемно-ориентированная функция, то уровень связности — функциональный; конец алгоритма. В противном случае перейти к пункту 2.
2. Если действия внутри модуля связаны, то перейти к пункту 3. Если действия внутри модуля никак не связаны, то перейти к пункту 6.
3. Если действия внутри модуля связаны данными, то перейти к пункту 4. Если действия внутри модуля связаны потоком управления, перейти к пункту 5.

3. Определения связности модуля

Алгоритм определения уровня связности модуля:

4. Если порядок действий внутри модуля важен, то уровень связности — информационный. В противном случае уровень связности — коммуникативный. Конец алгоритма.
5. Если порядок действий внутри модуля важен, то уровень связности — процедурный. В противном случае уровень связности — временной. Конец алгоритма.
6. Если действия внутри модуля принадлежат к одной категории, то уровень связности — логический. Если действия внутри модуля не принадлежат к одной категории, то уровень связности — по совпадению. Конец алгоритма.

3. Определения связности модуля



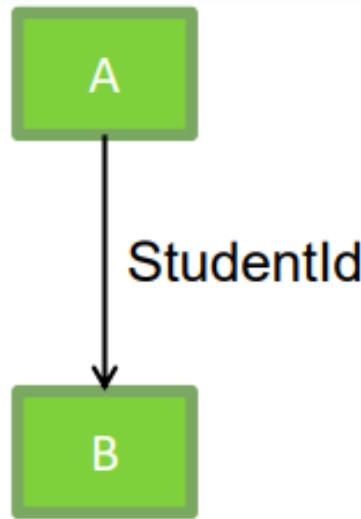
3. Сцепление модулей

Сцепление модулей (внешняя характеристика) – это мера взаимозависимости модулей по данным.

№ п/п	Сцепление	Степень сцепления
1	Независимое	0 (слабое сцепление)
2	По данным	1
3	По образцу	3
4	По общей области	4
5	По управлению	5
6	По внешним ссылкам	7
7	<u>По кодам (содержимому)</u>	<u>9 (сильное сцепление)</u>

3. Сцепление по данным

Модуль А вызывает модуль Б. Все входные и выходные параметры вызываемого модуля – простые элементы данных.



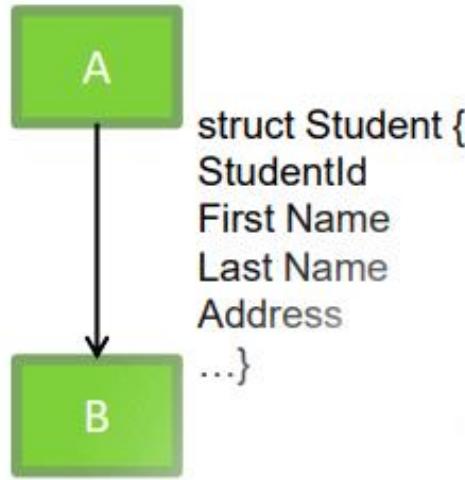
Взывающем модуле известны имя вызываемого модуля, а также типы и значения переменных, передаваемых как параметры. Вызываемый модуль озывающем может не содержать никакой информации.

В этом случае изменения в структуре данных в одном из модулей не влияют на другой модуль.

Модули со сцеплением по данным не имеют общей области данных (глобальных переменных).

3. Сцепление по образцу

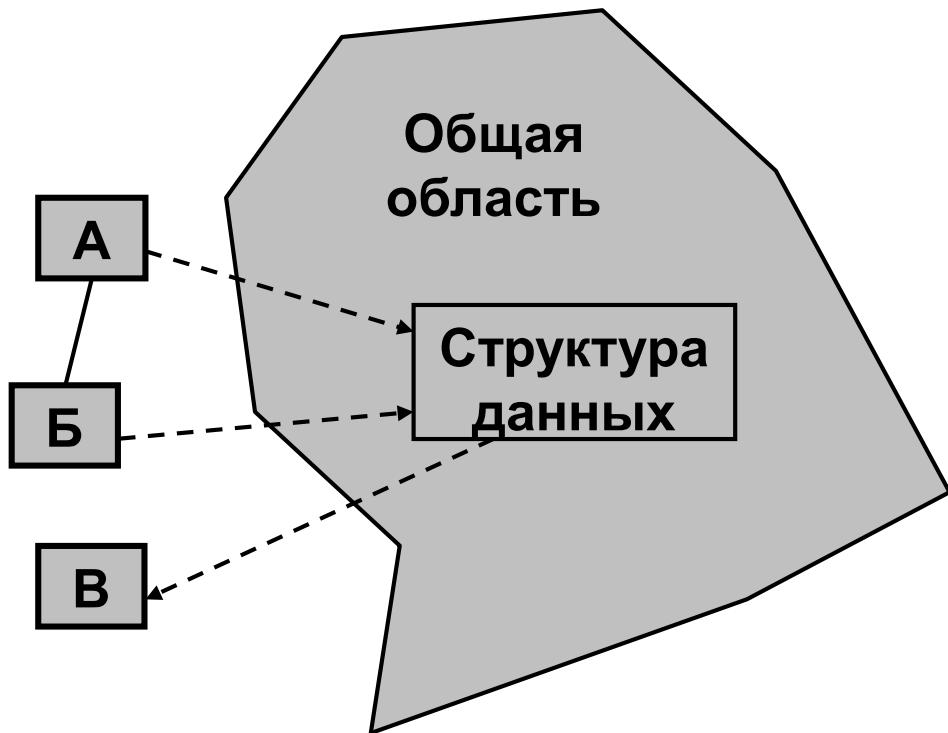
В качестве параметров используются структуры данных. Так как часто в структуре могут быть лишние, не используемые данные, такой тип связности менее предпочтителен.



Недостаток такого сцепления заключается в том, что оба модуля должны содержать информацию о внутренней структуре данных. Если модифицируется структура данных в одном из модулей, то необходимо модифицировать структуру данных и в другом. Следовательно, увеличивается вероятность появления ошибок при программировании и сопровождении программ.

3. Сцепление по общей области

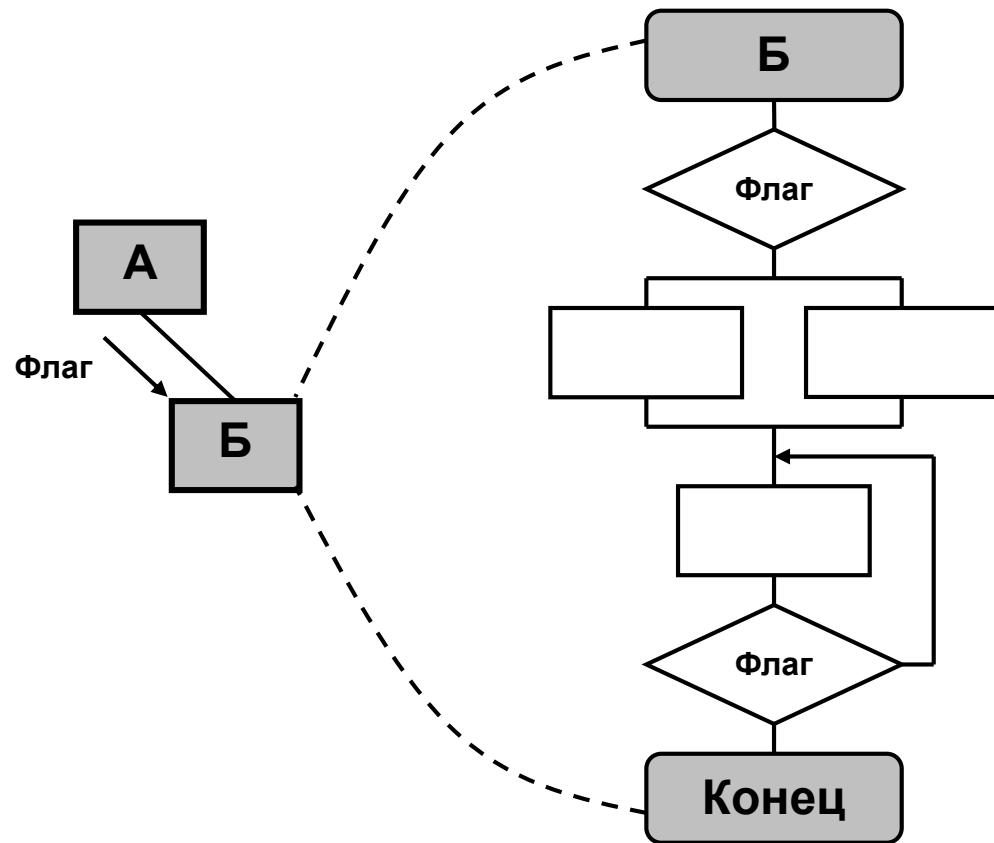
Модули разделяют одну и ту же глобальную структуру данных



В этом случае возможностей для появления ошибок при модификации структуры данных в одном из модулей намного больше, поскольку труднее определить модули, нуждающиеся в корректировке.

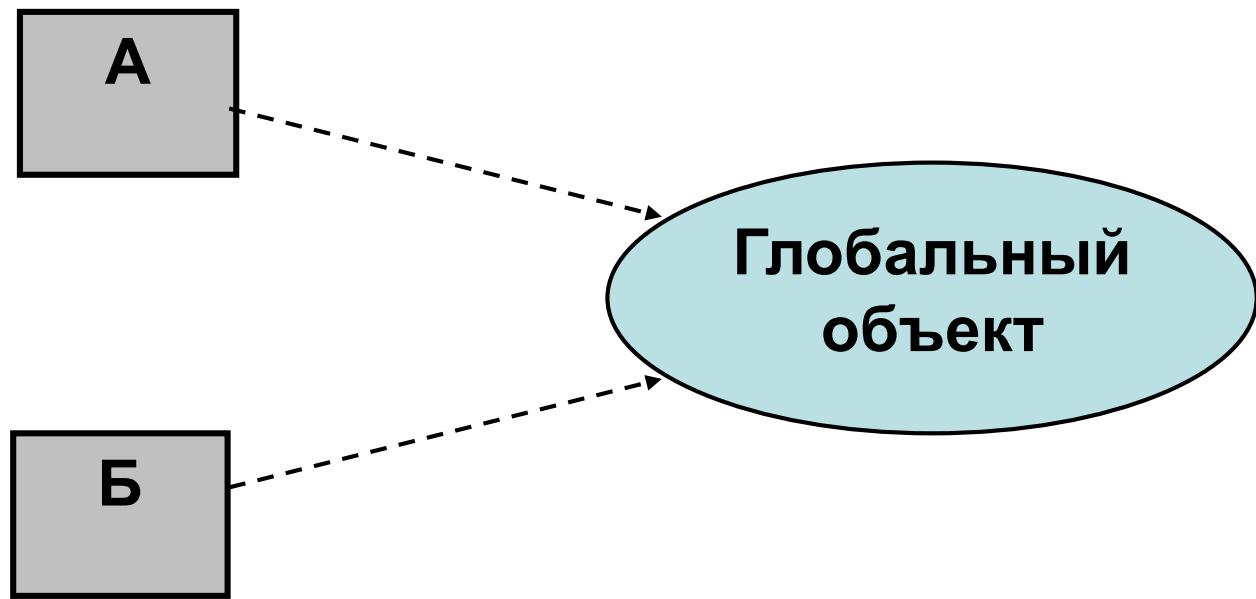
3. Сцепление по управлению

Модуль А явно управляет функционированием модуля Б, посылая ему управляющие данные. Таким образом, один из модулей содержит информацию о внутренних функциях другого.



3. Сцепление по внешним ссылкам

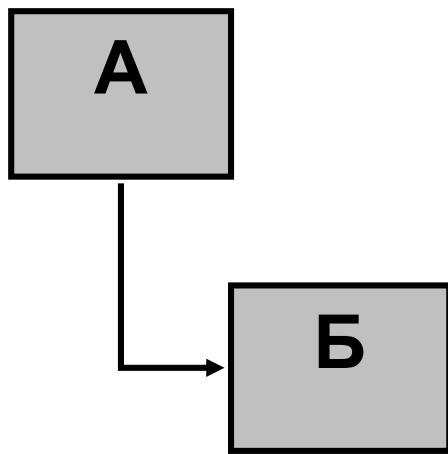
Модули А и Б ссылаются на один и тот же глобальный элемент данных. Таким путем осуществляется неявное управление функционированием другого модуля. Сцепление этого типа возникает, когда внутренние процедуры одного модуля оперируют с глобальными переменными другого модуля.



3. Сцепление по содержанию

Один модуль прямо ссылается на содержание другого модуля.

Например, для одного из модулей доступны внутренние области другого модуля без обращения к его точкам входа, то есть модули используют общий участок памяти с командами. Это сцепление возникает, когда модули проектируются как отдельные подпрограммы, путь через которые начинается в различных точках входа, но приводит к общему сегменту кодов.





Этап конструирования ПО

Вопросы:

1. Основы конструирования ПО
2. Управление конструированием

1. ОПРЕДЕЛЕНИЕ

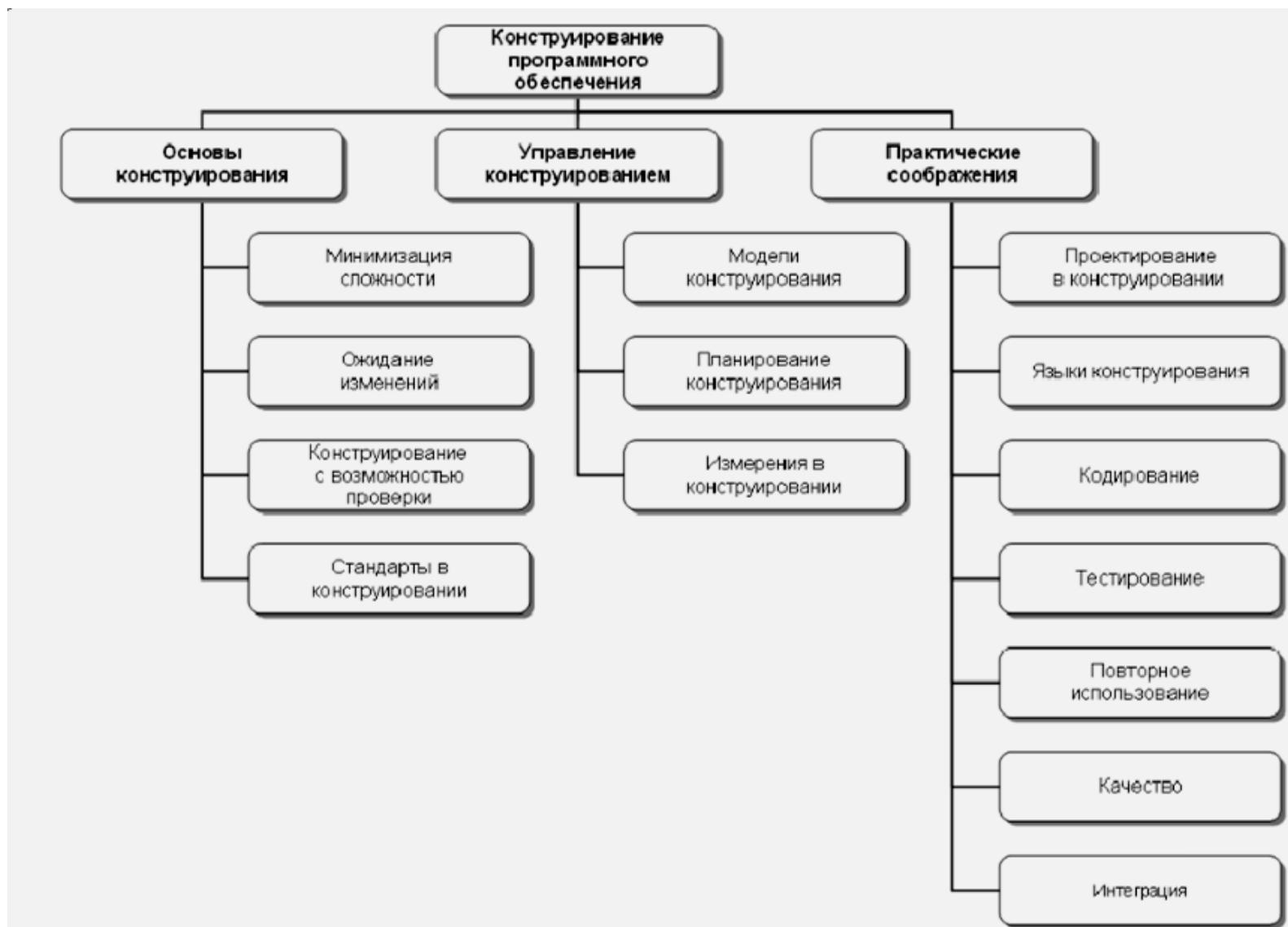
Конструирование программного обеспечения
заключается в создании рабочего программного
обеспечения посредством комбинации кодирования,
верификации (проверки), модульного тестирования,
интеграционного тестирования и отладки .

Программная инженерия. Проектирование программного обеспечения
(SWEBOK, Software Design)

1. СВЯЗЬ С ДРУГИМИ СТАДИЯМИ ЖЦ ПО



1. ОБЩАЯ СТРУКТУРА



1. ОСНОВЫ КОНСТРУИРОВАНИЯ

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

1. ОСНОВЫ КОНСТРУИРОВАНИЯ

1.1 Минимизация сложности

Обеспечивается с помощью :

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

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.

Пусть $C(x)$ – функция сложности решения проблемы x ,

$T(x)$ – функция затрат времени на решение проблемы x .

Для двух проблем p_1 и p_2 из соотношения $C(p_1) > C(p_2)$ следует, что $T(p_1) > T(p_2)$

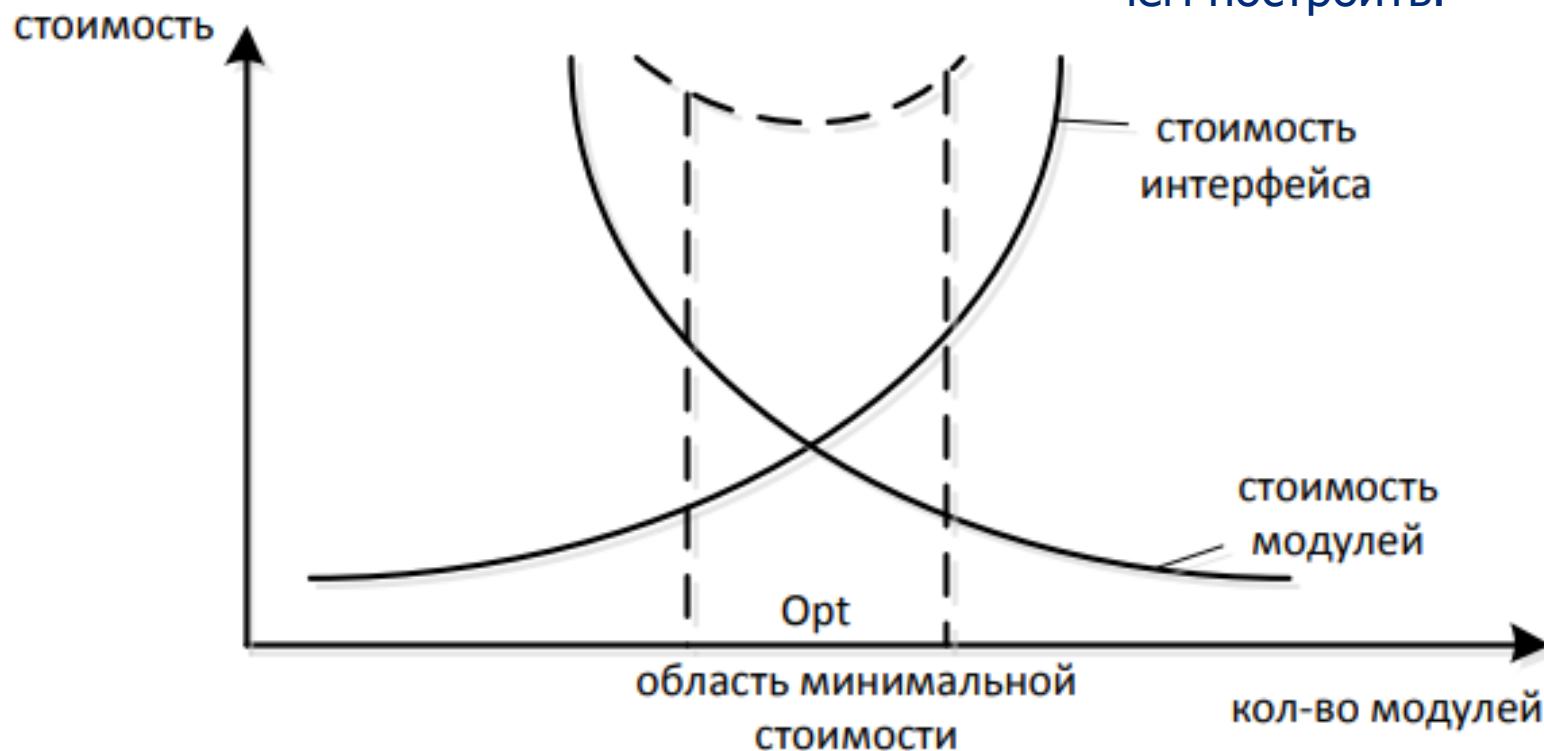
Модульный принцип разработки ПО

Затраты на модульность

$C(p_1 + p_2) > C(p_1) + C(p_2)$. Отсюда
 $T(p_1 + p_2) > T(p_1) + C(p_2)$.

Хороший модуль:

- снаружи проще, чем внутри;
- его проще использовать, чем построить.



Модульный принцип разработки ПО

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 operandов.

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.

программы

Модульный принцип разработки ПО

УГП - отображает поток управления программы.

Это **граф $G(V, A)$** , где $V(V_1, \dots V_m)$ – множество вершин (операторов), $A(A_1, \dots A_n)$ – множество дуг (управлений), соединяющих вершины.

Путь – последовательность вершин и дуг УГП, в которой любая дуга выходит из вершины V_i и приходит в вершину V_j

Ветвь – путь $(V_1, V_2, \dots V_k)$, где V_1 - либо первый, либо условный оператор, V_k - либо условный оператор, либо оператор выхода из программы, а все остальные операторы – безусловные. Ветви - линейные участки программы, их конечное число.

Число путей в программе может быть не ограничено (пути, различающиеся хотя бы числом прохождений цикла – разные).

Существуют реализуемые и нереализуемые пути в программе, в нереализуемые пути в обычных условиях попасть нельзя.

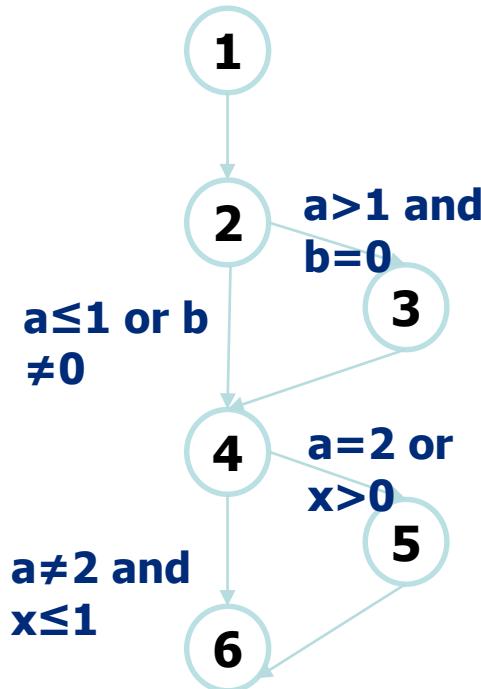
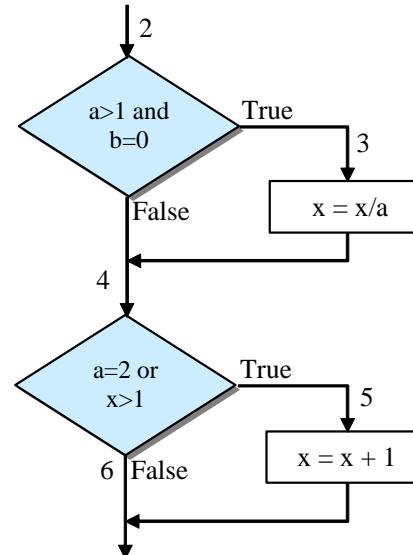
Условие при построении графа: каждая вершина достижима из начальной, и конечная вершина достижима из любой другой вершины

Модульный принцип разработки ПО

Управляющий граф программы: пример

Void m (float a, float b, float x)

1. {
2. If ($a > 1 \ \&\& \ b == 0$)
3. $x = /a;$
4. If ($a == 2 \ || \ x > 1$)
5. $x++;$
6. }



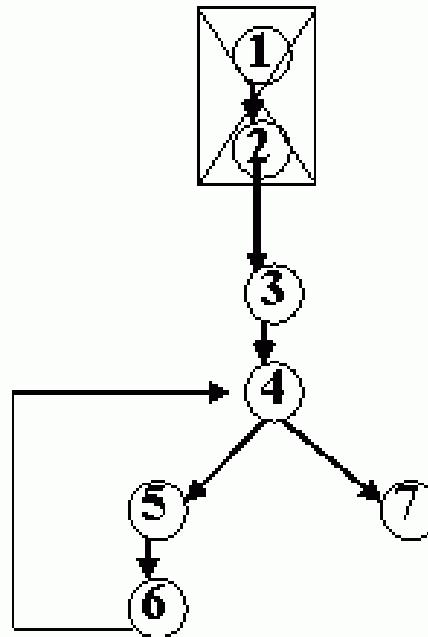
Пути: (1-2-3-4-5-6); (1-2-4-6); (1-2-4-5-6); (1-2-3-4-6)

Ветви: (2-3-4); (4-5-6); (2-4); (4-6)

Модульный принцип разработки ПО

Управляющий граф программы: пример

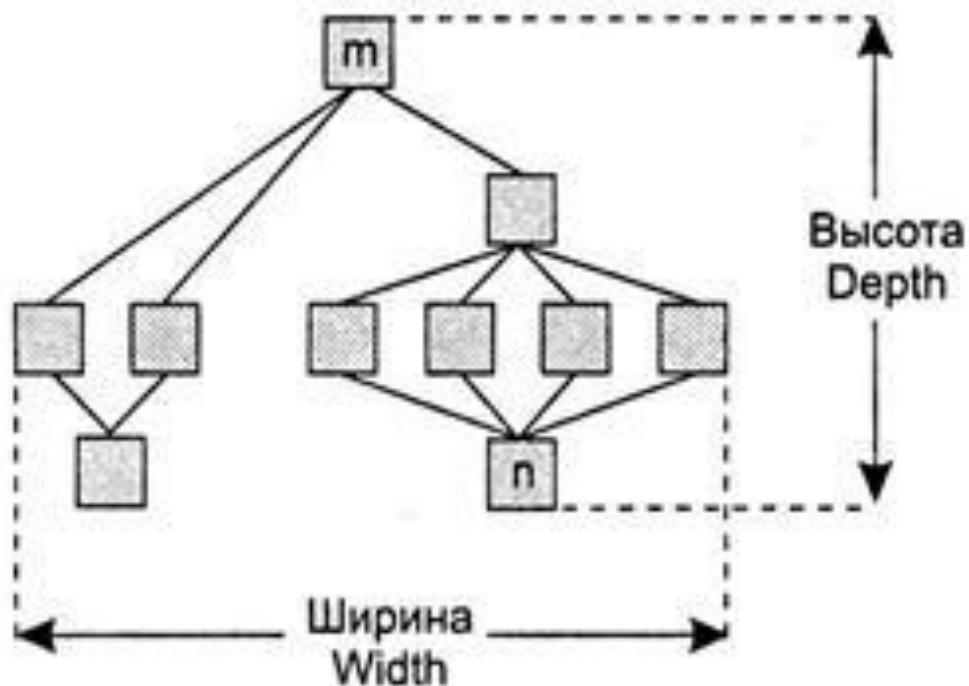
```
/* Функция вычисляет неотрицательную степень n числа x */
1 double Power(double x, int n) {
2 double z=1; int i
3 for (i=1;
4 n>=i;
5 i++)
6 {z = z*x;}
7 return z;}
```



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.

1.1 МИНИМИЗАЦИЯ СЛОЖНОСТИ



Пример иерархической структуры ПС

1. ОСНОВЫ КОНСТРУИРОВАНИЯ

Локальные характеристики модулей

- a) Коэффициент объединения по входу, $\text{Fan_in}(i)$ – количество модулей, которые прямо управляет i -тым;
- b) Коэффициент разветвления по выходу, $\text{Fan_out}(i)$ – количество модулей, которыми управляет i -тым модулем.

В примере $\text{Fan_in}(n) = 4$, $\text{Fan_out}(m) = 3$.

1.1 МИНИМИЗАЦИЯ СЛОЖНОСТИ

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.

1.1 МИНИМИЗАЦИЯ СЛОЖНОСТИ

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

1.1 МИНИМИЗАЦИЯ СЛОЖНОСТИ

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).

1.1 МИНИМИЗАЦИЯ СЛОЖНОСТИ

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.

1.1 МИНИМИЗАЦИЯ СЛОЖНОСТИ

Читабельность

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



Самодокументируемый код

1.1 МИНИМИЗАЦИЯ СЛОЖНОСТИ

Читабельность

Обеспечивать лёгкое покрытие кода тестами и отладку. Этому способствуют логичное именование и хороший стиль интерфейса и реализации.

Гарантировать лёгкость сопровождения.

Упрощать процесс внесения дальнейших изменений.

Обеспечивать устойчивость программы.

Обеспечивать возможность поддержки проекта несколькими разработчиками или целыми сообществами (особенно важно для проектов с открытым исходным кодом).

1.1 МИНИМИЗАЦИЯ СЛОЖНОСТИ

Читабельность

Обеспечивать лёгкое покрытие кода тестами и отладку. Этому способствуют логичное именование и хороший стиль интерфейса и реализации.

Гарантировать лёгкость сопровождения.

Упрощать процесс внесения дальнейших изменений.

Обеспечивать устойчивость программы.

Обеспечивать возможность поддержки проекта несколькими разработчиками или целыми сообществами (особенно важно для проектов с открытым исходным кодом).

1.1 МИНИМИЗАЦИЯ СЛОЖНОСТИ

Читабельность: как обеспечить

Использовать понятные имена классов, методов, функций и др. имя уже должно говорить, о чем эта переменная, функция, метод или класс, но делать имена длиннее 3-4 слов не стоит.

Для каждой отдельной задачи писать свой метод или функцию. В таком коде легче будет разобраться, о чем он и что выполняет конкретная функция или метод.

В меру комментировать код. Комментарии обязательны, особенно в тех местах, где могут возникнуть трудности с пониманием кода. Но с комментариями важно не переусердствовать, чтобы ваш код не превратился в художественный роман.

Планировать структуру программы. Прежде чем писать код, нужно его спланировать и обдумать.

Соблюдать переносы фигурных скобок и отступы.

Не использовать «магические числа».

1.1 МИНИМИЗАЦИЯ СЛОЖНОСТИ

Читабельность: как обеспечить

```
setTitle(text1); setSize(550,360); setVisible(true);
```

```
buttonSave.addActionListener(new ActionListener() {  
    @Override public void actionPerformed(ActionEvent e) {  
        buttonLogin();  
    } });
```

```
if (a) { if (b) { do_something(); } }
```

```
if (x) { var1 = y; } else { var1 = z; }
```

1.2. ОЖИДАНИЕ ИЗМЕНЕНИЙ

Существует два вида изменений:

- 1) Рефакторинг или усовершенствование кода программы

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.

1.2. ОЖИДАНИЕ ИЗМЕНЕНИЙ

Причины необходимости рефакторинга

- 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) несгруппированные данные.

1.2. ОЖИДАНИЕ ИЗМЕНЕНИЙ

Существует два вида изменений:

- 1) Рейнжиниринг - добавление новой функциональности

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 применением автоматизированных средств тестирования;
- 4) ограниченное применение сложных или тяжелых для понимания языковых структур.

1.3. КОНСТРУИРОВАНИЕ С ВОЗМОЖНОСТЬЮ ПРОВЕРКИ

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.

1.3. КОНСТРУИРОВАНИЕ С ВОЗМОЖНОСТЬЮ ПРОВЕРКИ

Модульное или юнит-тестирование

процесс, позволяющий проверить корректность отдельных модулей исходного кода программы.

Задачи

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

1.4 СТАНДАРТЫ КОНСТРУИРОВАНИЯ

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

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.

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.

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.

МЕТРИКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Согласно международному стандарту ISO 14598

Метрика - это количественный масштаб и метод, который может использоваться для измерения.

Метрика программного обеспечения (англ, software metric) - это мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификаций

РАЗМЕРНО — ОРИЕНТИРОВАННЫЕ МЕТРИКИ

Размерно-ориентированные метрики прямо измеряют программный продукт и процесс его разработки. Основываются на LOC-оценках.

LOC-оценка — это количество строк в программном продукте.

Кроме LOC к количественным характеристикам относят:

- количество пустых строк,
- количество комментариев,
- процент комментариев (отношение числа строк, содержащих комментарии к общему количеству строк, выраженное в процентах),
- среднее число строк для функций (классов, файлов),
- среднее число строк для модулей

РАЗМЕРНО – ОРИЕНТИРОВАННЫЕ МЕТРИКИ

Показатели:

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

РАЗМЕРНО – ОРИЕНТИРОВАННЫЕ МЕТРИКИ

Производительность =

Длина [тыс. LOC]

Затраты [чел.-мес.]

Качество =

Ошибки [Единиц]

Длина [тыс. LOC]

Удельная Стоимость =

Стоимость [Тыс. руб.]

Длина [LOC]

Документированность =

Страниц Документа

Длина [тыс. LOC]

РАЗМЕРНО – ОРИЕНТИРОВАННЫЕ МЕТРИКИ

Различают логические и физические строки кода.

количество «физических» строк кода – SLOC (используемые аббревиатуры LOC, SLOC, KLOC, KSLOC, DSLOC) – определяется как общее число строк исходного кода, включая комментарии и пустые строки (при измерении показателя на количество пустых строк, как правило, вводится ограничение – при подсчете учитывается число пустых строк, которое не превышает 25% общего числа строк в измеряемом блоке кода).

Количество «логических» строк кода – SLOC (используемые аббревиатуры LSI, DSI, KDSI, где «SI» — source instructions) – определяется как количество команд и зависит от используемого языка программирования.

РАЗМЕРНО – ОРИЕНТИРОВАННЫЕ МЕТРИКИ

Размерно-ориентированные метрики:

основаны на объективных данных

просты и легко вычислимы

Недостатки:

зависят от языка программирования

трудновыполнимы на начальной стадии проекта

не приспособлены к непроцедурным языкам программирования

Метод LOC является только оценочным методом, который надо принимать к сведению, но не опираться на его результаты в оценках

МЕТОДОЛОГИЯ ОЦЕНИВАНИЯ ФУНКЦИОНАЛЬНОГО РАЗМЕРА

Данная метрика ПО была разработана в противовес LOC в 70-х года прошлого столетия А. Дж. Альбректом для компании IBM .

Методика не зависит от языка программирования и среды разработки.

Вместо подсчета LOC-оценки рассматривается не размер, а функциональность или полезность продукта.

Сущность ее состоит в том, что программа разделяется на классы компонентов по формату и типу логических операций.

МЕТОДОЛОГИЯ ОЦЕНИВАНИЯ ФУНКЦИОНАЛЬНОГО РАЗМЕРА

Косвенно измеряют программный продукт и процесс его разработки.

Используется 5 информационных характеристик.

1. Количество внешних вводов.
2. Количество внешних выводов.
3. Количество внешних запросов.
4. Количество внутренних логических файлов.
5. Количество внешних интерфейсных файлов.

МЕТОДОЛОГИЯ ОЦЕНИВАНИЯ ФУНКЦИОНАЛЬНОГО РАЗМЕРА



Вводы, выводы и запросы относят к категории транзакция.
Транзакция — это элементарный процесс, различаемый пользователем и перемещающий данные между внешней средой и программным приложением.

Внешний ввод (EI, external inputs) — элементарный процесс, перемещающий данные из внешней среды в приложение.

Внешний вывод (EO, external outputs) — элементарный процесс, перемещающий данные, вычисленные в приложении, во внешнюю среду. Кроме того, в этом процессе могут обновляться внутренние логические файлы.

Внешний запрос (EQ,external inquiries) — элементарный процесс, работающий как с вводимыми, так и с выводимыми данными. Его результат — данные, возвращаемые из внутренних логических файлов и внешних интерфейсных файлов.

МЕТОДОЛОГИЯ ОЦЕНИВАНИЯ ФУНКЦИОНАЛЬНОГО РАЗМЕРА

Внутренние логические файлы (ILFs) — выделяемые пользователем логически связанные группы данных или блоки управляющей информации, которые поддерживаются внутри продукта.

Внешние интерфейсные файлы (EIFs) — выделяемые пользователем логически связанные группы данных или блоки управляющей информации, на которые ссылается продукт, но которые поддерживаются вне продукта.

МЕТОДОЛОГИЯ ОЦЕНИВАНИЯ ФУНКЦИОНАЛЬНОГО РАЗМЕРА

Каждой характеристике ставится в соответствие сложность, для этого назначается низкий, средний или высокий ранг, затем формируется числовая оценка ранга.

Имя характеристики	Ранг, сложность, количество			
	Низкий	Средний	Высокий	Итого
Внешние вводы	? x 3 =	? x 4 =	? x 6 =	= ?
Внешние выводы	? x 4 =	? x 5 =	? x 7 =	= ?
Внешние запросы	? x 3 =	? x 4 =	? x 6 =	= ?
Внутренние логические файлы	? x 7 =	? x 10 =	? x 15 =	= ?
Внешние интерфейсные файлы	? x 5 =	? x 7 =	? x 10 =	= ?
Общее количество				= ?

Примеры элементов данных

Информационная характеристика	Элементы данных
Внешние Вводы	Поля ввода данных, сообщения об ошибках, вычисляемые значения, кнопки
Внешние Выводы	Поля данных в отчетах, вычисляемые значения, сообщения об ошибках, заголовки столбцов, которые читаются из внутреннего файла
Внешние Запросы	Вводимые элементы: поле, используемое для поиска, щелчок мыши. Выводимые элементы — отображаемые на экране поля

Правила учета элементов данных GUI

Элемент данных	Правило учета
Группа радиокнопок	Так как в группе пользователь выбирает только одну радиокнопку, все радиокнопки группы считаются одним элементом данных
Группа флагков (переключателей)	Так как в группе пользователь может выбрать несколько флагков, каждый флагок считают элементом данных
Командные кнопки	Командная кнопка может определять действие добавления, изменения, запроса. Кнопка OK может вызывать транзакции (различных типов). Кнопка Next может быть входным элементом запроса или вызывать другую транзакцию. Каждая кнопка считается отдельным элементом данных
Списки	Список может быть внешним запросом, но результат запроса может быть элементом данных внешнего ввода

GUI для обслуживания клиентов может иметь поля Имя, Адрес, Город, Страна, Почтовый Индекс, Телефон, Email

Таким образом, имеется 7 полей или семь элементов данных. Восьмым элементом данных может быть командная кнопка (добавить, изменить, удалить). В этом случае каждый из внешних вводов Добавить, Изменить, Удалить будет состоять из 8 элементов данных (7 полей плюс командная кнопка).

Ранг и оценка сложности

Сложность данных определяется по следующим показателям:

DET (data element type) — неповторяющееся уникальное поле данных, например, Имя Клиента — 1 DET; Адрес Клиента (индекс, страна, область, район, город, улица, дом, корпус, квартира) — 9 DET's

RET (record element type) — логическая группа данных, например, адрес, паспорт, телефонный номер.

РАНГ И ОЦЕНКА СЛОЖНОСТИ

Внешних входов

Ссылки на файлы (RET)	Элементы данных(DET)		
	1-4	5-15	>15
0-1	Низкий (3)	Низкий (3)	Средний (4)
2	Низкий (3)	Средний (4)	Высокий (6)
>2	Средний (4)	Высокий (6)	Высокий (6)

Внешних выходов

	1-4	5-19	>19
0-1	Низкий (4)	Низкий (4)	Средний (5)
2-3	Низкий (4)	Средний (5)	Высокий (7)
>3	Средний (5)	Высокий (7)	Высокий (7)

Внешних запросов

Ссылки на файлы	Элементы данных		
	1-4	5-19	>19
0-1	Низкий (3)	Низкий (3)	Средний (4)
2-3	Низкий (3)	Средний (4)	Высокий (6)
>3	Средний (4)	Высокий (6)	Высокий (6)

Расчет количества функциональных указателей FP (Function Points)

$$FP = \text{Общее количество} \times (0,65 + 0,01 \times \sum_{i=1}^{14} F_i),$$

где F_i — коэффициенты регулировки сложности.

Каждый коэффициент может принимать следующие значения:
0 - нет влияния, 1 - случайное, 2 - небольшое, 3 - среднее, 4 -
важное, 5 - основное.

МЕТОДОЛОГИЯ ОЦЕНИВАНИЯ ФУНКЦИОНАЛЬНОГО РАЗМЕРА

$$\text{Производитель} = \frac{\text{Функций Указатель}}{\text{Затраты}} \left[\frac{\text{FP}}{\text{чел. - мес}} \right];$$

$$\text{Качество} = \frac{\text{Ошибки}}{\text{Функций Указатель}} \left[\frac{\text{Единиц}}{\text{FP}} \right];$$

$$\text{Удельная стоимость} = \frac{\text{Стоимость}}{\text{Функций Указатель}} \left[\frac{\text{Тыс. \$}}{\text{FP}} \right];$$

$$\text{Документированность} = \frac{\text{Страниц Документа}}{\text{Функций Указатель}} \left[\frac{\text{Страниц}}{\text{FP}} \right].$$

МЕТОДОЛОГИЯ ОЦЕНИВАНИЯ ФУНКЦИОНАЛЬНОГО РАЗМЕРА

Язык программирования	Количество операторов на один FP
Ассемблер	320
C	128
Фортран	106
Паскаль	90
C++	64
Java	53
Ada95	49
Visual Basic	32
Visual C+	34
Delphi(Pascal)	29
Smalltalk	22
HTML3	15
LISP	64
Prolog	64