Технологии программирования

- Программное обеспечение := компьютерные программы и соответствующая документация.
- Программирование := процесс создания программного обеспечения.
- Программное обеспечение := {любительское, промышленное:={«коробочное» ПО, сделанное на заказ}}.
- Промышленное ПО := заказчик + команда (фирма разработчик) + бюджет (деньги) + сроки.
- Современное ПО СЛОЖНО:
  - Сложность реального мира. Взаимоотношения между заказчиком, пользователями и разработчиками.
  - Трудность управления процессами разработки. Разработка – коллективный процесс. Программирование – процесс творческий.
  - Гибкость ПО. В отличие от других областей возможно все написать «с нуля», не всегда есть готовые «кирпичи-компоненты».
  - Проблемы описания поведения больших дискретных систем. (Задачи, которые ставятся для решения программному обеспечению сложны).
- **Необходимы технологии по созданию ПО : = совокупность процессов, ведущих к созданию или развитию ПО.**
- **Технологий разработано много, но все они содержат следующие базовые процессы:**

Базовые процессы разработки ПО (существуют в любой технологии)

- **Разработка спецификации ПО**: = определяет все функции и действия, которые будет выполнять разрабатываемое ПО.
- **Проектирование** : = на основе спецификации разработка архитектуры системы.
- **Реализация** := создание ПО (кодирование, написание документации и т.д.).
- **Аттестация**: = верификация и аттестация. Разработанное ПО должно быть аттестовано на соответствие заказчику.
- **Эволюция ПО**: = дальнейшая модификация ПО в соответствии с требованиями заказчика.

Технологии отличаются друг от друга последовательностью и порядком применения и проведения базовых процессов. Для систематизации технологий выделяются модели. Модель : = абстрактное представление процесса

PS:Знать наизусть даже на два.

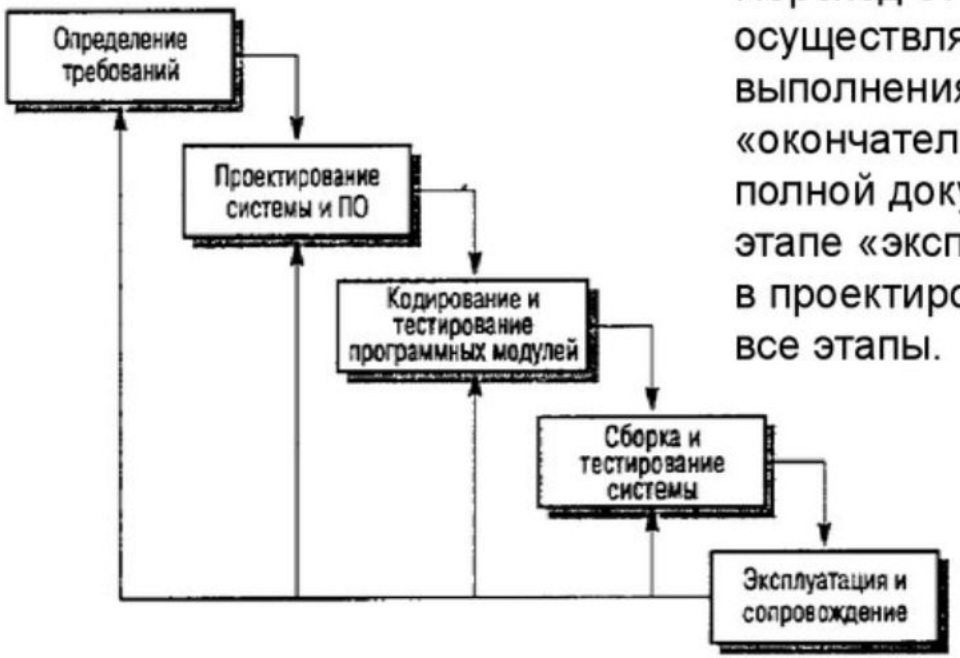
Модели процесса создания ПО (жизненного цикла)

- Каскадная (водопадная модель)
- Эволюционная модель: = {подход пробных разработок, прототипирование}
- Разработка на основе ранее созданных компонент
- Спиральная модель

Жизненный цикл : = это совокупность процессов, протекающих в период от момента принятия решения о создании ПО до его полного выхода из строя. Жизненный цикл шире чем модели создания ПО.

Модели показывают возможные подходы к разработке ПО, на практике могут применяться любые «комбинации» этих подходов. Например, какой-то этап в спиральной модели может быть осуществлен с применением каскадной модели.

Каскадная модель



Переход от одного этапа к другому осуществляется только после окончательного выполнения предыдущего. В понятие «окончательное выполнение» входит написание полной документации по данному этапу. Если на этапе «эксплуатации» обнаруживаются ошибки в проектировании, то процесс снова проходит все этапы.

-Не гибкое разделение процесса создания на этапы
-Определяющее значение имеют решения принятые на ранних этапах. Если возникают ошибки на ранних этапах, то приходится повторять все с начала.
-Классический подход требует параллельной документации каждого процесса.
+ Хорошо отражает и структурирует процесс создания ПО
+ Используется при проектировании систем, где системные требования четко определены заранее.

Эволюционная модель разработки

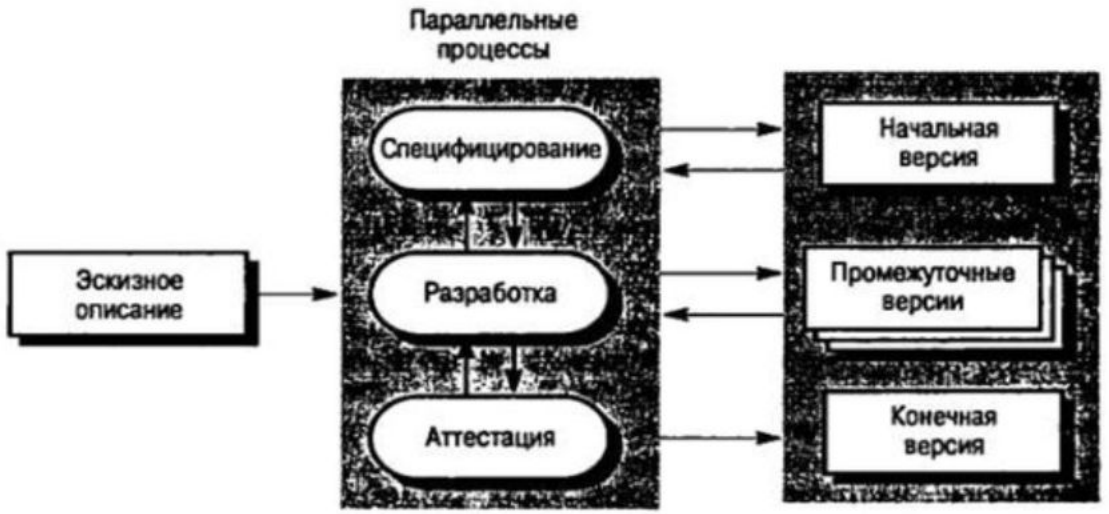


Рис. 3.2. Эволюционная модель разработки

Идея – разрабатывается первоначальная версия ПО, которая передается на испытание пользователям, затем она дорабатывается с учетом мнение пользователей, получается промежуточная версия и т.д. Пока окончательная версия не будет удовлетворять пользователя. Процессы специфицирование, разработки, аттестации выполняются параллельно при постоянном взаимном обмене информацией.

- Многие этапы не документированы.

- Система получается плохо структурированной (иногда).

- Часто требуются специальные средства и технологии разработки.

+ Более эффективен, чем каскадная модель если требования заказчика могут меняться в процессе разработки.

+Спецификация разрабатывается постепенно, по мере того как заказчик осознает и формулирует задачи, которое должно решать ПО.

Разработка на основе ранее созданных компонент

Рис. 3.5. Разработка ПО с повторным использованием ранее созданных компонентов

!!! После анализа функциональности компонентов возможна модификация требований. Так как не все требования заказчика в полном объеме могут быть реализованы существующими компонентами.

+ Процесс построения системы заключается в компоновке готовых компонентов.
+ Уменьшается срок создания ПО.
+ Уменьшается стоимость ПО из-за использования готовых частей.
+ Требуется тестировать только сборку, не требуется тестировать готовые компоненты.

-На данный момент «рынок» компонентов окончательно не сформировался. Не понятно, где брать и искать эти готовые компоненты.
-Не вся функциональность может быть реализована на основании готовых компонентов.

Спиральная модель

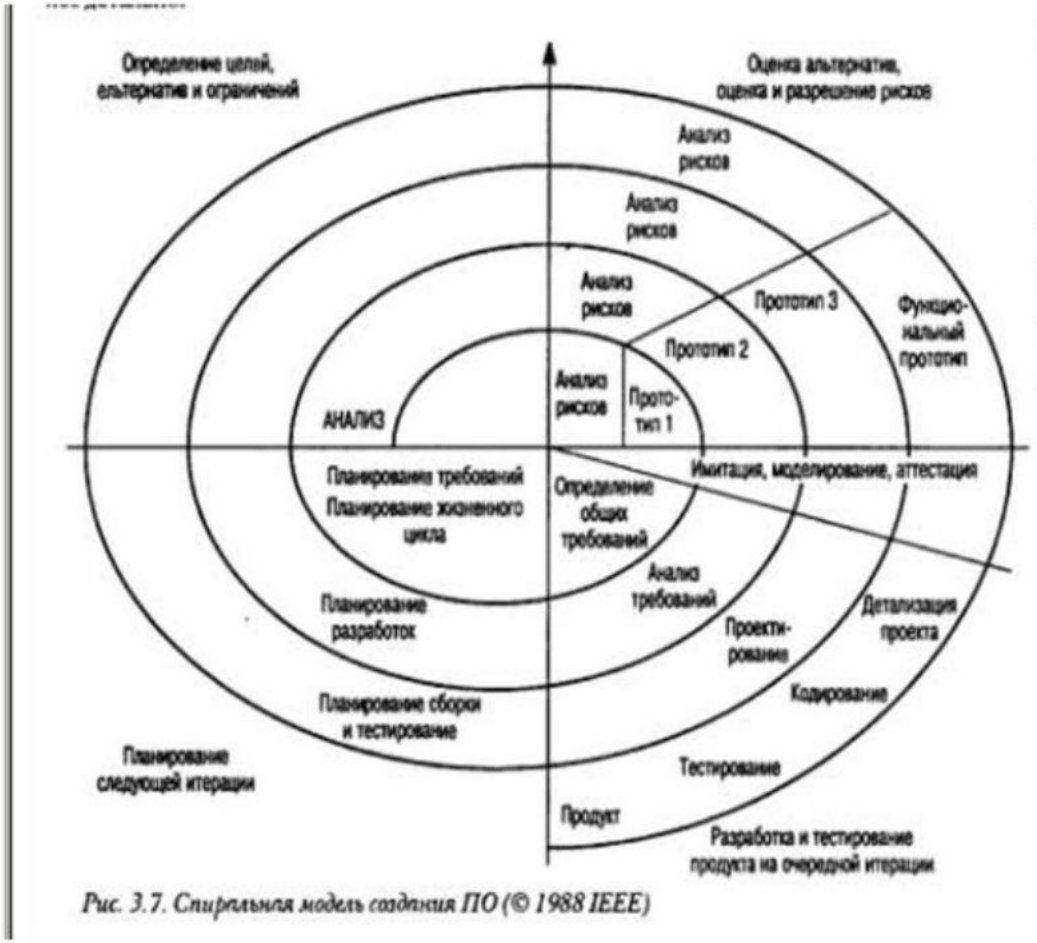


Рис. 3.7. Спиральная модель создания ПО (© 1988 IEEE)

Каждый виток спирали:

1.Определение целей
2.Оценка и разрешение рисков
3.Разработка и тестирование
4.Планирование.

Нет четких этапов как в других моделях. Эта модель может включать в себя любые другие модели.

Разработка спецификации ПО

•Предварительное исследование - оценивается степень удовлетворенности пользователей существующем ПО, бюджетные ограничения на разработку нового ПО, его экономическая эффективность.

•Формирование и анализ требований – изучение существующих аналогичных систем, обсуждение будущего ПО с заказчиками и пользователями, анализ задач и т.д. Может включать разработку нескольких моделей системы и ее прототипов, что помогает сформировать функциональные требования к системе.

•Специфицирование требований – перевод всей информации в «бумажный» документ.

•Утверждение требований – утверждение сформированных требований заказчиком.

Требования к программному обеспечению.

- Пользовательские требования – описание на естественном языке (плюс диаграммы) функций, выполняемых системой, и ограничений, накладываемых на систему.
- Системные требования – детализированное описание системных функций и ограничений. Основа для заключения контракта между покупателями и заказчиком.
- Проектная системная спецификация – дополняет и детализирует системные требования. Основа для более детализированного проектирования системы. (Обычно составляется для разработчиков системы)

Таблица 5.1. Пользовательские и системные требования

Пользовательские требования
1. ПО должно предоставить средство доступа к внешним файлам, созданным в других программах.
Спецификация системных требований
1.1. Пользователь должен иметь возможность определять тип внешних файлов.
1.2. Для каждого типа внешнего файла должно иметься соответствующее средство, применимое к этому типу файлов.
1.3. Внешний файл каждого типа должен быть представлен соответствующей пиктограммой на дисплее пользователя.
1.4. Пользователю должна быть предоставлена возможность самому определять пиктограмму для каждого типа внешних файлов.
1.5. При выборе пользователем пиктограммы, представляющей внешний файл, к этому файлу должно быть применено средство, ассоциированное с внешними файлами данного типа.

Пример пользовательских и системных требований.

Требования к программному обеспечению

- Функциональные требования – перечень сервисов, которые должна выполнять система, должно быть указано, как система реагирует на входные данные, как ведет в определенных ситуациях и т.д.
- Нефункциональные требования – описывает характеристики системы и ее окружения, а не поведение системы. Перечень ограничений, накладываемых на действия, выполняемые системой.
  - Требования к продукту – эксплуатационные свойства программного продукта (требования к производительности системы, объемы необходимой памяти, надежности, удобства эксплуатации и т.д.)
  - Организационные требования – отображают политику и организационные процедуры заказчика и разработчика ПО (стандарты разработки ПО, сроки изготовления, требования к документации и т.д.)
  - Внешние требования – учитываются факторы, внешние по отношению к ПО и его разработке, например, взаимодействие системы с другими системами, этические правила и т.д.
- Требования к предметной области – характеризует ту предметную область, где будет эксплуатироваться система. Делятся на функциональные и нефункциональные.

Способы записи системных требований

Спецификации системных требований часто пишутся на естественных языках. Но применение естественных языков подразумевает, что те, кто пишет спецификацию и те, кто ее читают, одни и те же слова и выражения понимают одинаково. Поэтому существуют следующие методы записи системных требований:

Структурированный естественный язык	Сокращенная форма естественного языка, предназначенная для написания спецификаций. Стандартные формы и шаблоны.
Языки описания программ	Использование специальных структурированных языков. Например, псевдокода или PDL (program description language) и т.д.
Графические нотации	Графический язык, используемый для описания функциональных требований диаграммы и блок-схемы. Например, стандарты IDEF0,3 и UML.
Математические спецификации.	Система нотаций, основанная на математических концепциях. Формализованная однозначная, лишенная двусмысленности запись системных требований. Но она понятна далеко не всем заказчикам.

Структурированный естественный язык.

Пример спецификации системного требования:

Врезка 5.8. Спецификация системного требования, использующая стандартную форму

Эклипс/АРМ/Средства/DE/RD/3.5.1

Функция. Добавление структурных элементов в схему.

Описание. Добавление структурных элементов в существующую схему системной архитектуры. Пользователь выбирает тип структурного элемента и его местоположение. После вставки в схему структурный элемент становится выделенным (текущим структурным элементом). Пользователь определяет местоположение элемента путем перемещения курсора по области схемы.

Входные данные. Тип элемента, позиция элемента, идентификатор схемы.

Источники входных данных. Тип элемента и позиция элемента задаются пользователем, идентификатор схемы получен из базы данных проекта.

Выходные данные. Идентификатор схемы.

Пункт назначения. База данных проекта. Идентификатор схемы помещается в базу данных проекта по завершении выполнения данной функции.

Для выполнения функции требуется схема, определенная входным идентификатором схемы.

Предусловие. Схема открыта и отображается на экране пользователя.

Постусловие. Схема, за исключением вставки нового структурного элемента, не изменяется.

Побочные эффекты. Нет.

Спецификация: Эклипс/АРМ/Средства/DE/RD/3.5.1

Математическая спецификация. Формальные спецификации для параллельных систем. Сети Петри.

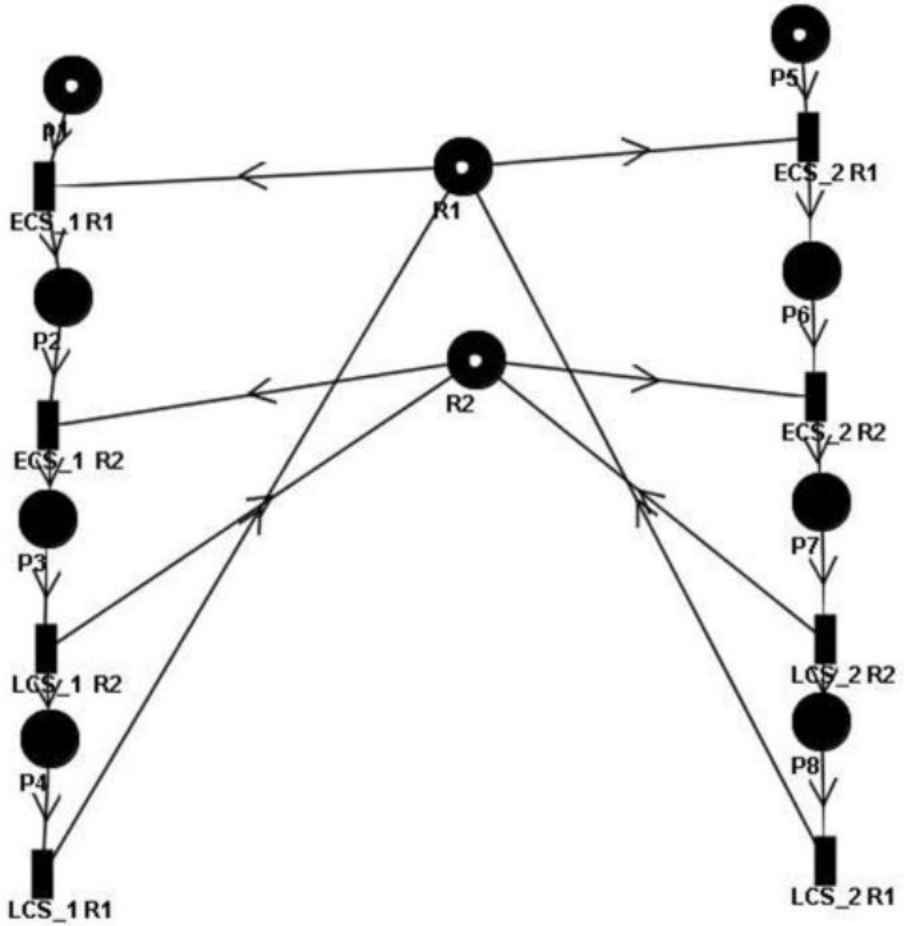


Рис.1 Сеть Петри, специфицирующая данный код

Потоки использующие два разделяемых ресурса.

```
CriticalSection R1,R2;
void ThreadExecute1()
{   EnterCriticalSection &R1;
EnterCriticalSection &R2;
Выполнить действия с R1 и R2
LeaveCritialSection &R2
LeaveCriricalSection &R1}
Void ThreadExecute2()
{EnterCriticalSection
&R1;EnterCriticalSection &R2;
Выполнить действия с R1 и
R2LeaveCritialSection
&R2LeaveCriricalSection &R1}
Int Main() {
Запустить поток 1
Запустить поток 2
}
```

Математическая спецификация. Продолжение.

- Сети Петри могут представляться с помощью графов, как показано на предыдущем примере, а могут иметь формальную запись, например, в виде протоколов флаговых и пусковых функций.
- Для формально записанной спецификации можно проводить анализ соответствующими методами. Спецификация записанная формальным языком может быть проанализирована (проверена, исследована) автоматически. Этим и интересна формальная спецификация.
- Таблица флаговых и пусковых функций для сети Петри рис.1

Ψt(ECS_1 R1) = R1*P1*¬P2
Φt+1(ECS_1 R1):R1=0; P1=0; P2=1

Ψt(ECS_1 R2) = R2*P2*¬P3
Φt+1(ECS_1 R2): R1=0; P1=0; P2=1
(аналогично можно построить и для всех остальных переходов)

Ψ – пусковая функция, задает правило срабатывания перехода

Φ - флаговая функция, задает состояние флагов после срабатывания перехода

Иногда флаговые и пусковые функции называют одним термином: ассинхронный протокол.

Модели систем

- Распространенная **методика документирования системных требований** является **построение ряда моделей системы**. Модели используют графическое представления, показывающие как решение задачи, для которой создается система, так и разрабатываемой системы. Модели являются связующим звеном между процессом анализа и проектированием системы.
- Модели показывают систему в различных аспектах:
  - Внешнее представление, когда моделируется окружение или рабочая среда системы.
  - Описание поведения системы, когда моделируется ее поведение
  - Описание структуры системы, когда моделируется архитектура системы или структуры данных, обрабатываемые системой.
- Типы системных моделей, которые могут создаваться в процессе анализа систем
  - Модель обработки данных. Последовательность обработки данных в системе.
  - Композиционные модели (диаграммы «сущность-связь»). Как одни сущности связаны с другими.
  - Архитектурная модель. Основные подсистемы из которых строится система.
  - Классификационная модель. (диаграмма классов, показывает какие объекты имеют общие характеристики).
  - Модель «стимул-ответ» . Диаграммы изменения состояний показывают, как система реагирует на внутренние и внешние события.

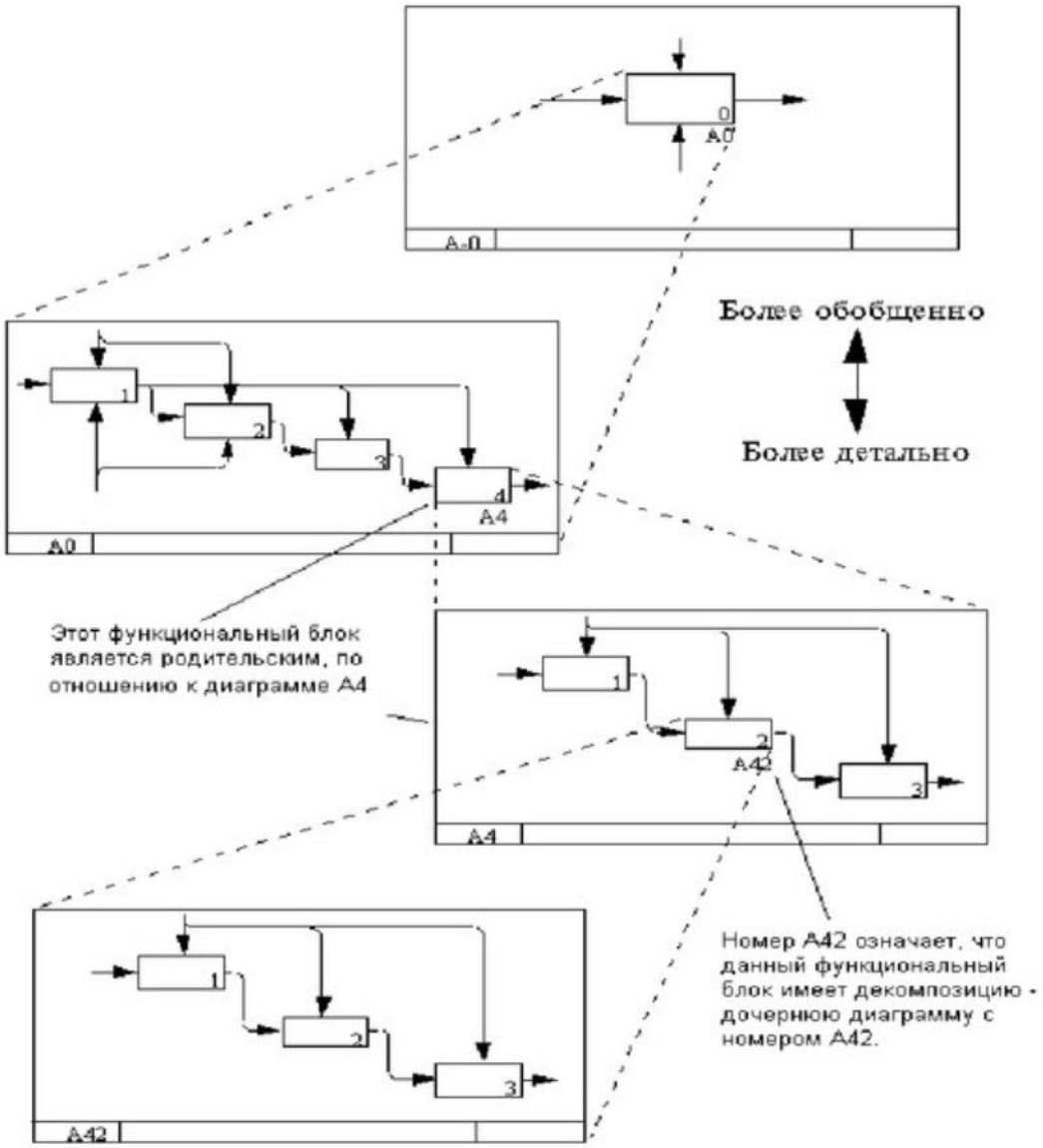
Процесс анализа (неформальное введение).

- Процесс анализа - необходим на этапе разработки системных требований, чтобы понять какие задачи должно решать создаваемое ПО и правильно построить модель системы.
- **Строить модель** и **документировать** можно **как угодно**, в любых понятных терминах и изображениях. Но чтобы вас **понимали другие** лучше использовать **готовые языки (методы) построения моделей**. Почти всегда язык определяет также и технологию моделирования.
- Структурный анализ - проводится с целью исследования статических характеристик системы путем выделения в ней подсистем и элементов различного уровня и определения отношений и связей между ними.
  - Описание функциональной структуры системы (метод SADT)
  - Последовательность выполняемых действий (метод IDEF3)
  - Передача информации между функциональными процессами (метод DFD)
  - Отношения между данными (метод ER. Модель «сущность-связь»).
- Объектно-ориентированный анализ (OOA)– методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области. (UML).

SADT – Structured Analysis and Design Technique.

- IDEF0 (Icam Definition)- методология функционального моделирования. С помощью наглядного графического языка IDEF0, изучаемая система предстает перед разработчиками и аналитиками в виде набора взаимосвязанных функций (функциональных блоков - в терминах IDEF0). Как правило, моделирование средствами IDEF0 является первым этапом изучения любой системы.
- **Основные понятия IDEF0:**
  - **Activity Box – функциональный блок**. Каждая из четырех сторон функционального блока имеет своё определенное значение (роль), при этом:
    - Верхняя сторона имеет значение “Управление” (Control);
    - Левая сторона имеет значение “Вход” (Input);
    - Правая сторона имеет значение “Выход” (Output);
    - Нижняя сторона имеет значение “Механизм” (Mechanism).
  - **Arrow – интерфейсная дуга**. Отображает элемент системы, который обрабатывается функциональным блоком или оказывает иное влияние на функцию, отображенную данным функциональным блоком.
  - **Декомпозиция -** разбиение процесса на составляющие функции. Позволяет представлять модель системы в виде иерархической структуры отдельных диаграмм, что делает ее менее перегруженной и легко усваиваемой. Модель IDEF0 всегда начинается с представления системы как единого целого – одного функционального блока с интерфейсными дугами, простирающимися за пределы рассматриваемой области.
  - **Глоссарий** Для каждого из элементов IDEF0: диаграмм, функциональных блоков, интерфейсных дуг существующий стандарт подразумевает создание и поддержание набора соответствующих определений, ключевых слов, повествовательных изложений и т.д., которые характеризуют объект, отображенный данным элементом.

Пример декомпозиции структурных диаграмм. На примере IDEF0.

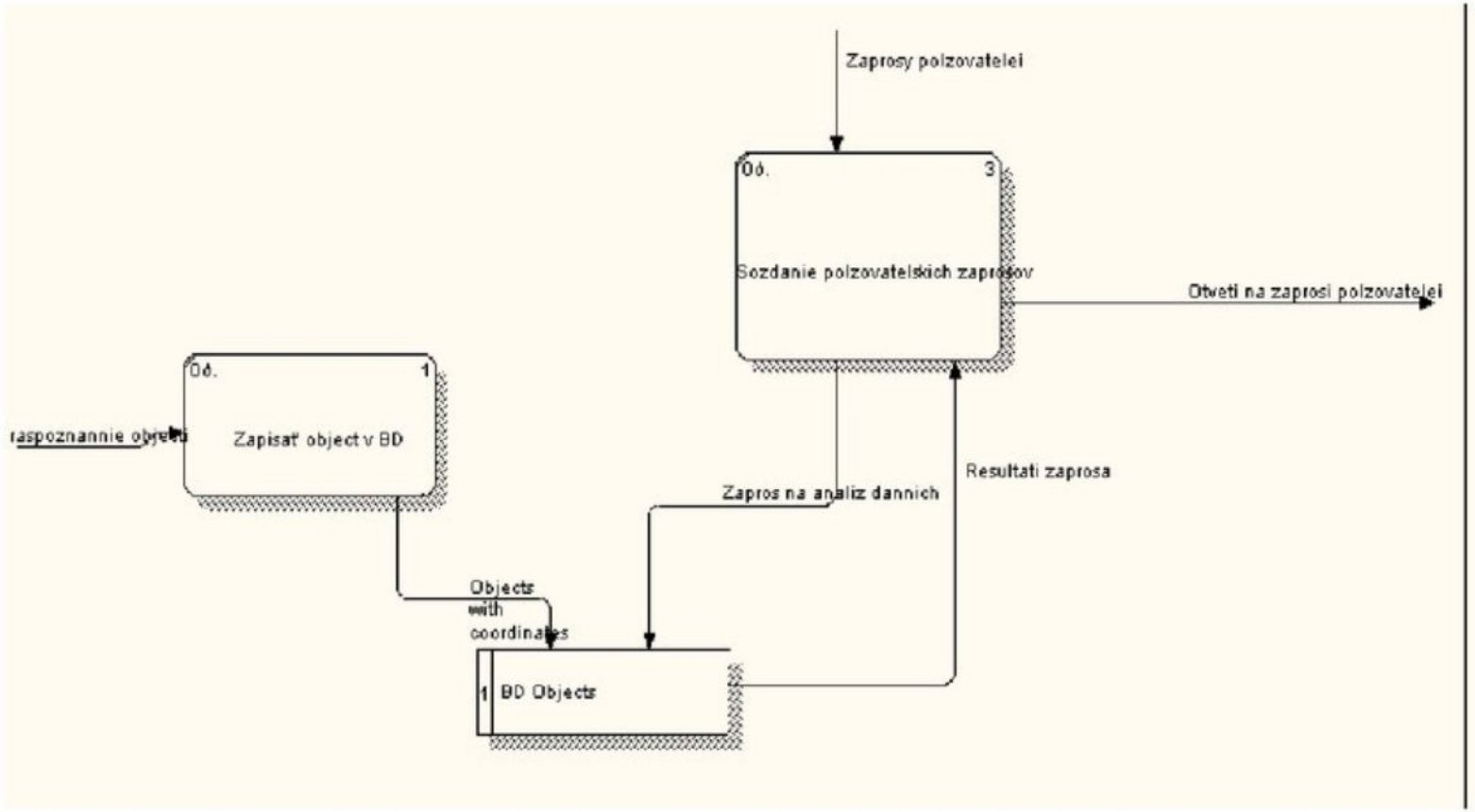


IDEF3

- IDEF3 является стандартом документирования технологических процессов, происходящих на предприятии, и предоставляет инструментарий для наглядного исследования и моделирования их сценариев. **Сценарием** (Scenario) называется описание последовательности изменений свойств объекта, в рамках рассматриваемого процесса.
- Два типа диаграмм:
  - **PFDD** (Process Flow Description Diagrams) Диаграмма Описания Последовательности Этапов Процесса.
    - **Функциональный блок** (UOB – Unit of Behavior)
    - **Перекрестки (Junction).** Используются для отображения логики взаимодействия стрелок (потоков) при слиянии и разветвлении или для отображения множества событий, которые могут или должны быть завершены перед началом следующей работы
      - Перекрестки слияния (Fan-in junction)
      - Перекрестки разветвления (Fan-out junction)
      - 5 типов перекрестков (AND (синхр. и асинхр.), OR (синхр. и асинхр.), XOR)
  - **OSTN** (Object State Transition Network). Диаграмма Состояния Объекта и его Трансформаций в процессе. Состояния объекта отображаются окружностями, а их изменения направленными линиями. Каждая линия имеет ссылку на соответствующий функциональный блок UOB, в результате которого произошло отображаемое ей изменение состояния объекта

Data Flow Diagram (DFD)

- Диаграммы потоков данных 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. Модель системы определяется как иерархия потоков данных, описывающих процесс преобразования информации от входа в систему до выдачи пользователю.



ER – диаграммы. IDEF1X.

- Метод ER (Entity Relationship) «сущность-связь» основан на выделении в предметной области «сущностей» и отражении взаимосвязи между этими сущностями.
  - **Сущность** (entity) - это "предмет" рассматриваемой предметной области, который может быть идентифицирован некоторым способом, отличающим его от других "предметов". Конкретные человек, компания или событие являются примерами сущности.
  - **Связь** (relationship) - это некоторое отношение между двумя и более сущностями, отражающее то, как они участвуют в общей деятельности, взаимодействуют друг с другом, совместно используются некоторой другой сущностью и т. д.

- Стандарт IDEF1 – моделирование предметной области с помощью ER-диаграмм.
- IDEF1X – метод проектирования реляционных баз данных.(ERWin), основанный на ER-диаграммах.
- CASE средства (например, ERWin) позволяют спроектировать реляционную структуру базы данных в виде диаграмм. При этом поддерживается два уровня представлений: логический и физический. На основании диаграмм генерируется скрипт (SQL/DDL). Существует Forward Engineering – генерация скрипта на основании модели, Reverse Engineering – обратная генерация диаграммы на основании базы данных.

Пример диаграммы IDEF1X. Логическая модель.