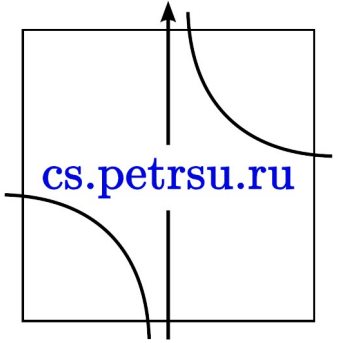


Организация процесса разработки ПО

Глава №7

§1 Что такое ТППО

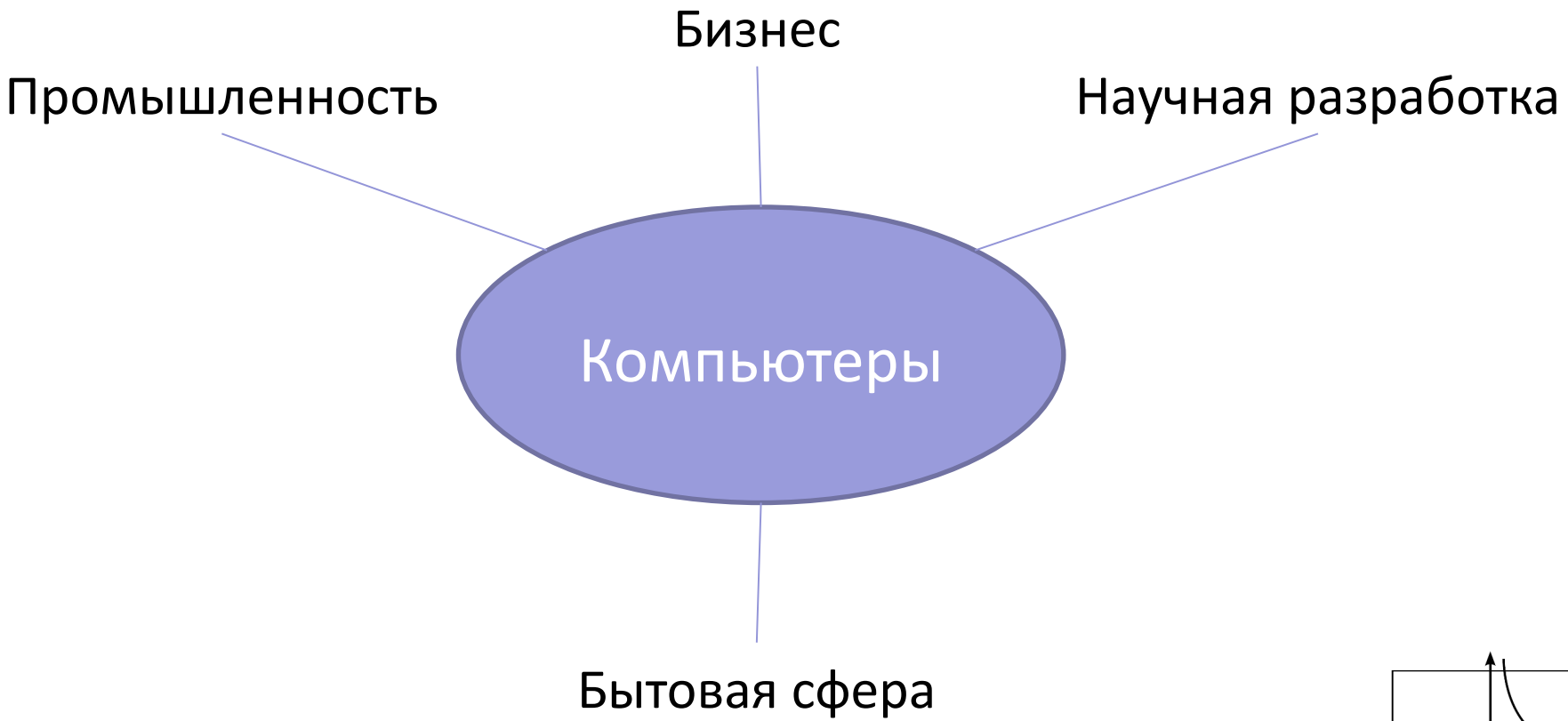
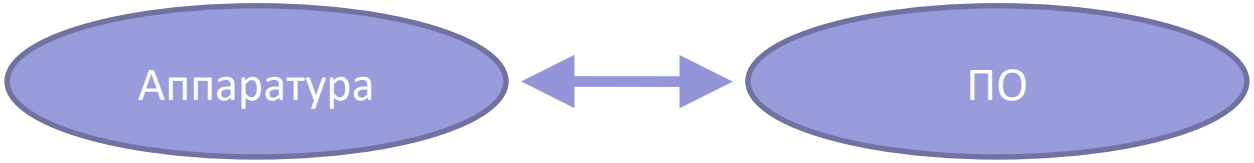


Рис.1. Огромная роль компьютеров в жизни человечества

cs.petrsu.ru

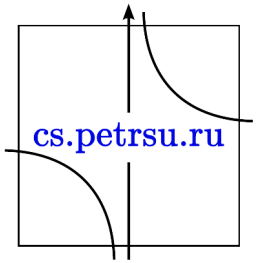


	Вычислительные и др. устройства	Программы, документация, конфигурация
	Рост мощности	Рост сложности
	Стоимость падает	Стоимость растет
60-е	Единичные ЭВМ. Все ок.	
70-е	Несколько ЭВМ	Кризис Командная работа ↓ Сложность и стоимость растет ↓
80-е	Сети	Распределенная работа
90-е	Массовость	• Ориентация на обычного пользователя • Часть больших систем(ИС) ↓ Нет чистого программирования

cs.petrsu.ru

Проблемы

- Аппаратная сложность опережает умение строить ПО, использование по максимому возможностей аппаратуры
- Умение строить ПО отстает от новых требования к этому ПО
- Возможностям по эксплуатации ПО угрожает низкое качество существующего ПО и его разработки




Новые требования
Аппаратная сложность
Отстает
Опережает
Возможность эксплуатации
Умение строить ПО
Качество ПО
Угрожает

cs.petrsu.ru

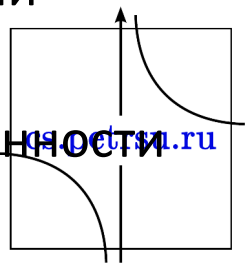
ТППО - определения

- СЭС: (от др.-греч. τέχνη — искусство, мастерство, умение; λόγος — «слово», «мысль», «смысл», «понятие»), совокупность методов обработки, изготовления, изменения состояния, свойств, формы сырья, материала или полуфабриката, осуществляемых в процессе производства продукции. Задача Т. Как науки – выявление физических, химических, механических и др. закономерностей с целью определения и использования на практике наиболее эффективных и экономичных производственных процессов.
- Фатрелл, Шафер (Engineering = технология производства)

  Метод – образ действий, средства или процесс для выполнения действий

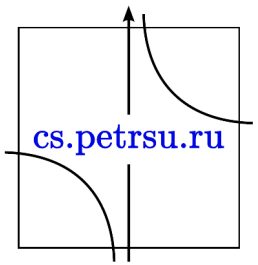
  Инструмент – орудие или механизм для выполнения работы или решения задачи

  Технология – способ применения научного знания в промышленности или бизнесе.



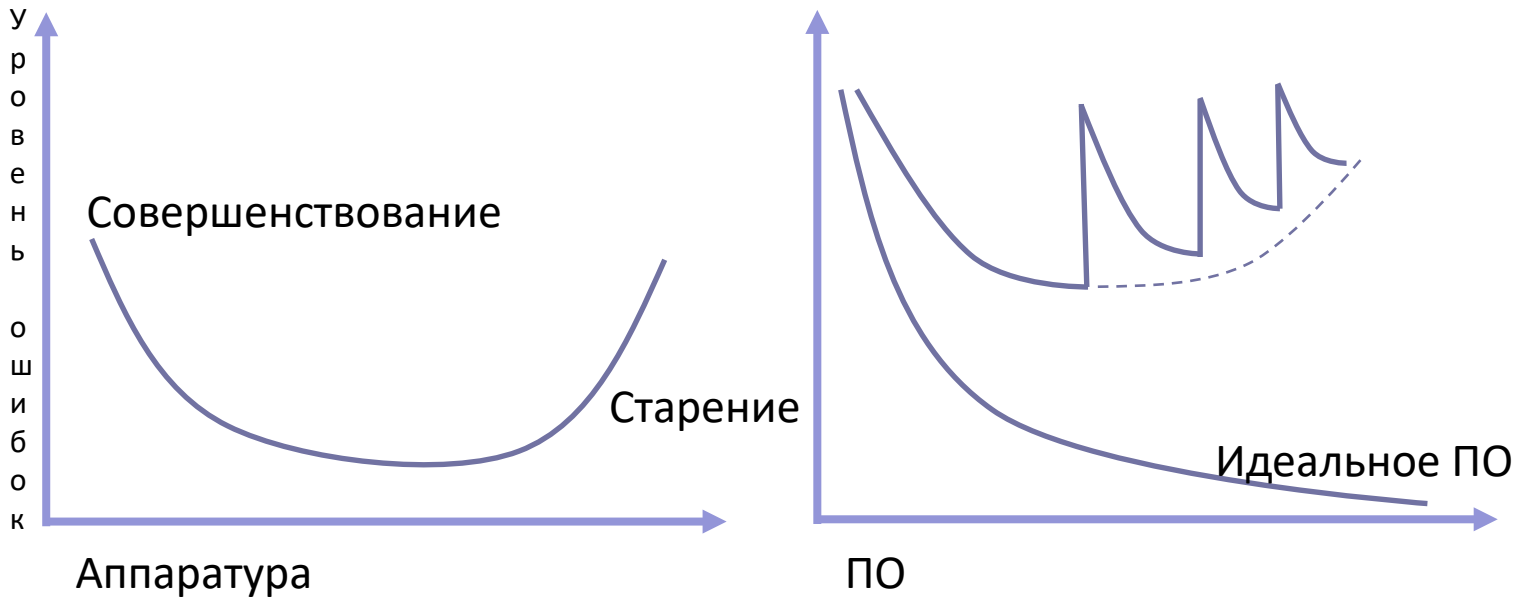
Основные аспекты ТППО

1. ТППО – это не кодирование/программирование и не алгоритмика
   - Сложность проекта
   - Командная(групповая) работа
   - Документирование
   - Работа с заказчиком/пользователем
2. Фундамент ТППО (истоки)
   - Классические инженерные дисциплины
   - Математика
   - Новые методы, связанные со спецификой ПО
3. Цель ТППО
   - Создание экономичного ПО, которое надежно и эффективно работает на реальных компьютерах
     a. Ограничения(время, ресурсы, стоимость, качество, …)
     b. Теория и практика – не всегда согласуются
     c. Человеческий фактор
4. Задачи ТППО
   - Планирование и оценка проекта
   - Анализ системных и программных требования
   - Проектирование алгоритмов и структур данных
   - Кодирование, тестирование
   - Сопровождение



Специфика ПО

- ПО разрабатывается(производится), а не изготавливается в классическом смысле
- ПО не стареет



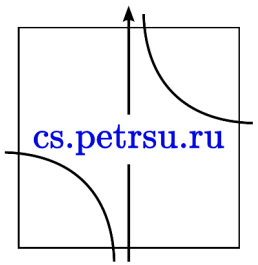
Casetools

- Computer Aided Software Engineering
  - компьютерная разработка программного обеспечения
  - программная инженерия с компьютерной поддержкой
- Средства обеспечения автоматизированной и автоматической поддержки методов

cs.petrsu.ru

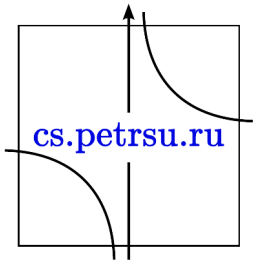
Компоненты CASE

- Обеспечение управления проектом
  - работа с расписанием
  - распределение работ
- Управление конфигурацией ПС (среда сборки)
- Управление требованиями
- Проектирование
  - функциональное
  - объектно-ориентированное
  - варианты использования
- Инструменты отслеживания
  - переходы требования в проектные решения
  - проектных решений в программный код
- Обеспечение тестирования
- Обеспечение технической поддержки



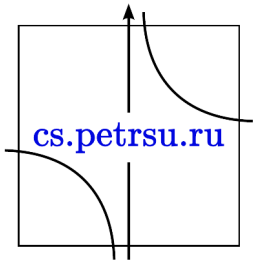
Достоинства CASE

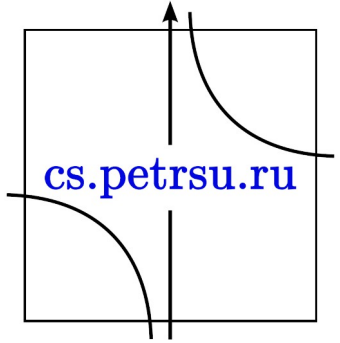
- Повышение качества
- Ускорение процесса проектирования и разработки
- Создание прототипов и оценка результата
- Освобождение от “рутины”
- Поддержка технологий повторного использования программ
- Поддержка развития и сопровождение разработки



Сравнение традиционного и CASE подходов

Традиционная разработка	CASE технологии
Основные усилия на кодирование и тестирование	Основные усилия на анализ и проектирование
Бумажная спецификация	Быстрое итеративное прототипирование
Ручное кодирование	Автоматическая кодогенерация
Ручное документирование	Автоматическая генерация документации
Ручное тестирование	Автоматический контроль

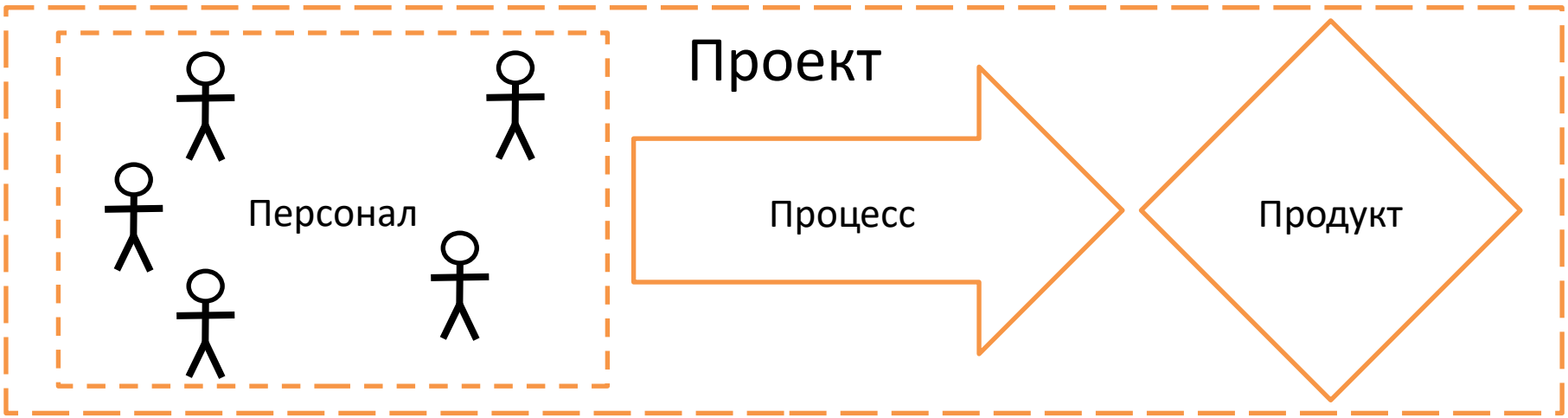




Процесс создания ПО

Глава №1

§ 1 ППП – персонал, продукт, процесс, проект



- Персонал – подбор, управление / человеческий фактор
- Продукт – назначение, характеристики / работа с заказчиком
- Процесс – набор действий, приводимых к создания продукта / обеспечение качества
- Проект – единственный известный способ управления сложностью

cs.petrsu.ru

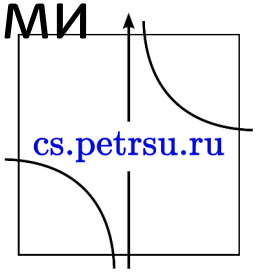
Статистика

- 2010:

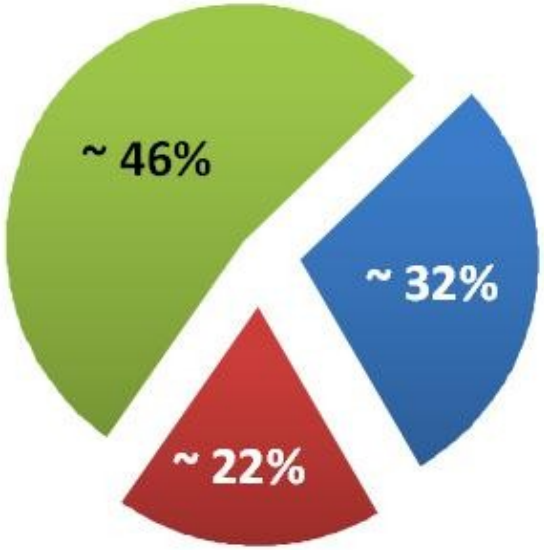
  32% программных проектов являются успешными, 44% являются спорными (имеющими перерасход средств, превышение бюджета, другие недостатки), а 24% являются провальными.

- 1994:

  16% проектов были успешными, 53% спорными и 31% неудачными



Неудачные проекты
(превышен бюджет, или сроки, или не достигли всех поставленных целей)



Успешные проекты

Провалившиеся проекты
(деньги затрачены впустую)

Исследование The Standish Group за 2008-2015 гг

https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf

cs.petrsu.ru

§2 ТППО – многоуровневая технология

Процесс склеивает методы и средства воедино

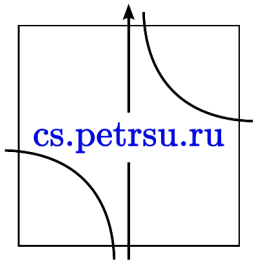
- Порядок применения методов и утилит
- Формирование отчетов
- Контроль и координация активности

cs.petrsu.ru

Функциональные этапы процесса

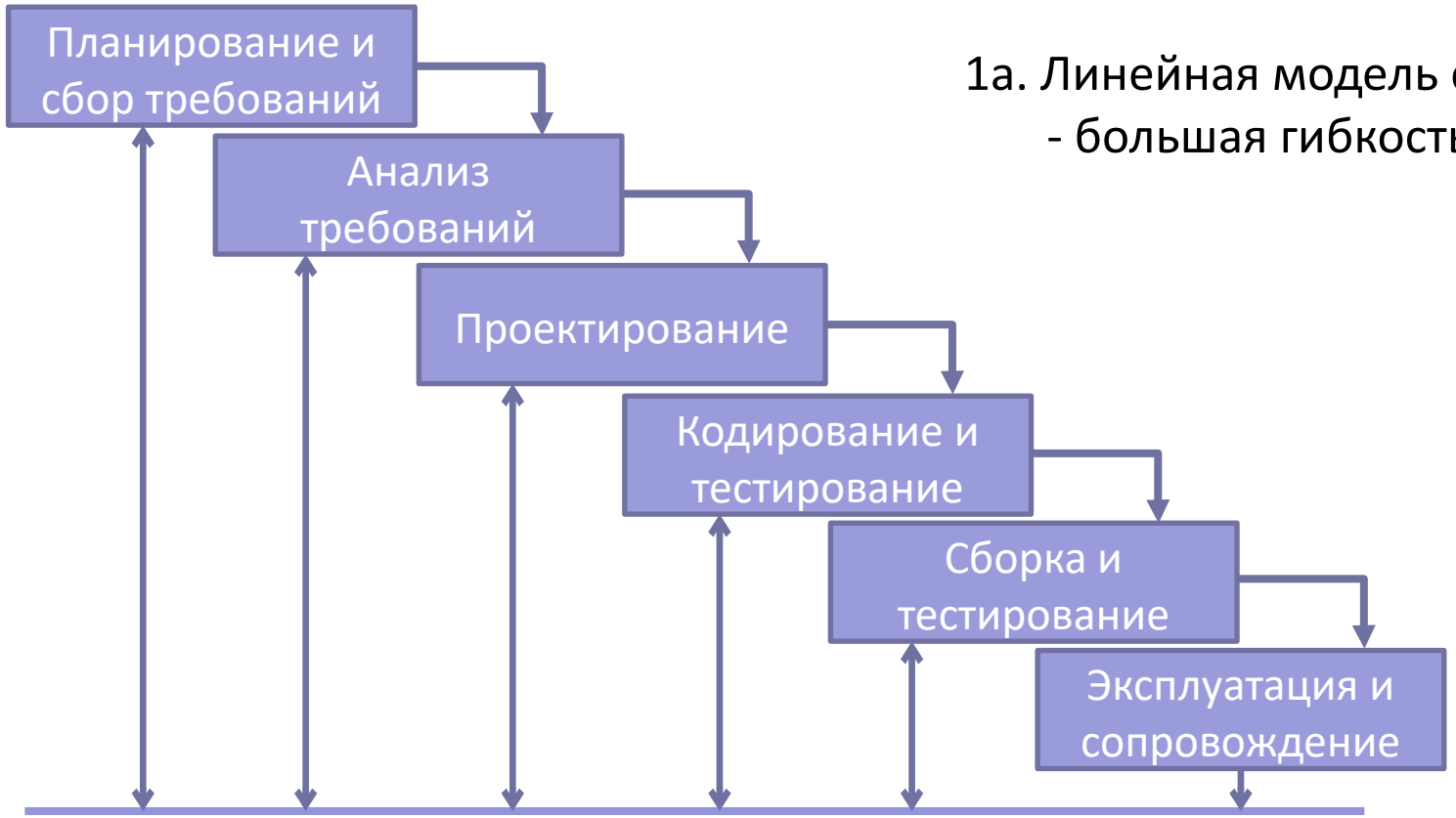
1. Этап спецификации (ЧТО)
   - Предметная область
   - Планирование
   - Анализ требований
2. Этап разработки (КАК)
   - Проектирование
   - Реализация
   - Аттестация
3. Этап сопровождения (Изменения)
   - Исправление (ошибки)
   - Адаптация
   - Расширение

Методы обеспечивают решения этих задач



§3 Модели процесса

1. Линейная (водопад, классическая модель, классический жизненный цикл) – наследство от старых инженерных дисциплин

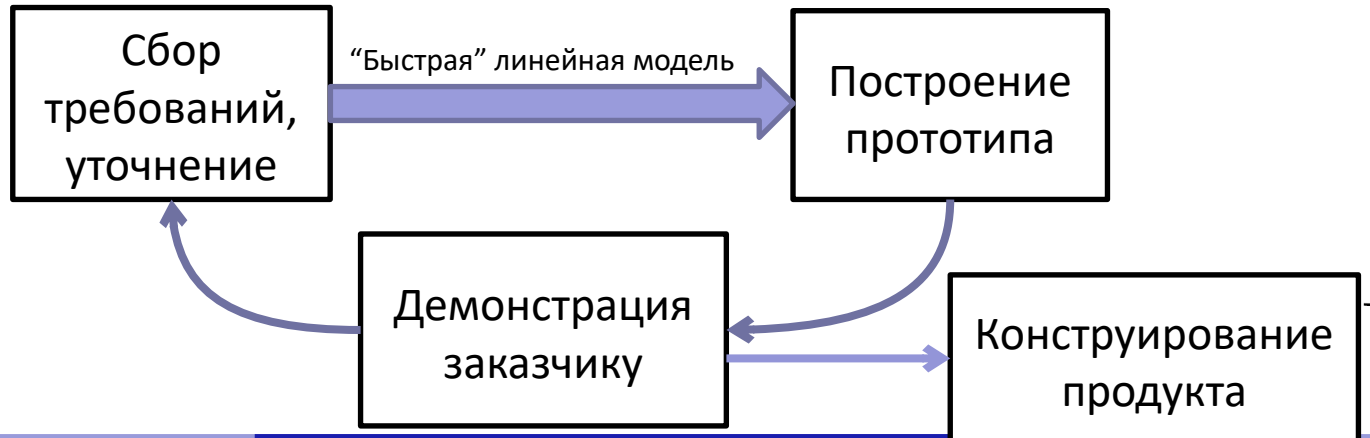


1а. Линейная модель с возвратами - большая гибкость

cs.petrsu.ru

2. Прототипирование (макетирование)

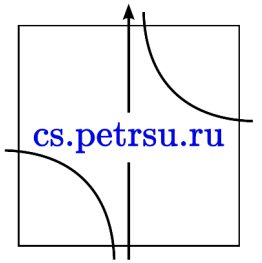
- поэтапное уточнение требований заказчика
- существенную часть спецификации требований составляет прототип ПС
  - Все требования заказчик сразу не может сформулировать (эволюционное прототипирование)
  - Непонятно заранее, возможно ли создание требуемого ПО или насколько оно полезно (экспериментальное прототипирование)



- Преимущества
  - Заказчик быстро получает представление о ПО
  - Активная обратная связь с заказчиком

- Недостатки
  - Заказчик может принять прототип за продукт
  - Разработчик может принять прототип за продукт

    т.о. и заказчик, и разработчик могут спутать прототип с реальной системой



3. Компонентная модель

- Повторное использование программных модулей (компонент)
- Наличие базы компонент

Анализ компонентов – поиск компонентов и их соответствие требованиям

Модификация требований – максимально использовать возможностей отобранных компонентов, альтернативные решения

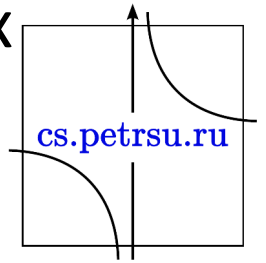
Проектирование системы – структура система в соответствии с функциональными возможностями компонентов. Для недоступных компонентов – проект нового ПО

Разработка и сборка – непосредственное создание системы

cs.petrsu.ru

- Преимущества
  - Уменьшение времени создания
  - Уменьшение количества создаваемых программ
  - Уменьшение стоимости
  - Надежность

- Недостатки
  - Соответствие требованиям заказчика
  - Невозможность влиять на появление новых версий компонент



4. RAD – Rapid Application Development (быстрая разработка приложений)

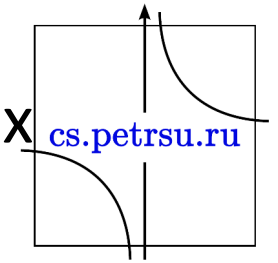
- Чрезвычайно короткий цикл разработки
- Высокоскоростная адаптация линейной модели, в которой быстрая разработка достигается использованием компонентной модели

Условия:

- Требования хорошо понятные и чёткие
- Предметная область ограничена (объем работ)
- Соответствующая база компонент в наличии

Тогда разработка полнофункциональной системы возможна за очень короткий срок.

Применяется для приложений в информационных системах



Стадии RAD:

1. Бизнес моделирование

   Потоки информации между бизнес-функциями (какая информация управляет бизнес-процессами – ищется ответ на вопрос ”Что за информация генерируется, Кто её генерирует? Куда идет? Кто её обрабатывает?”)

2. Моделирование данных

   Объекты данных (представление информации)

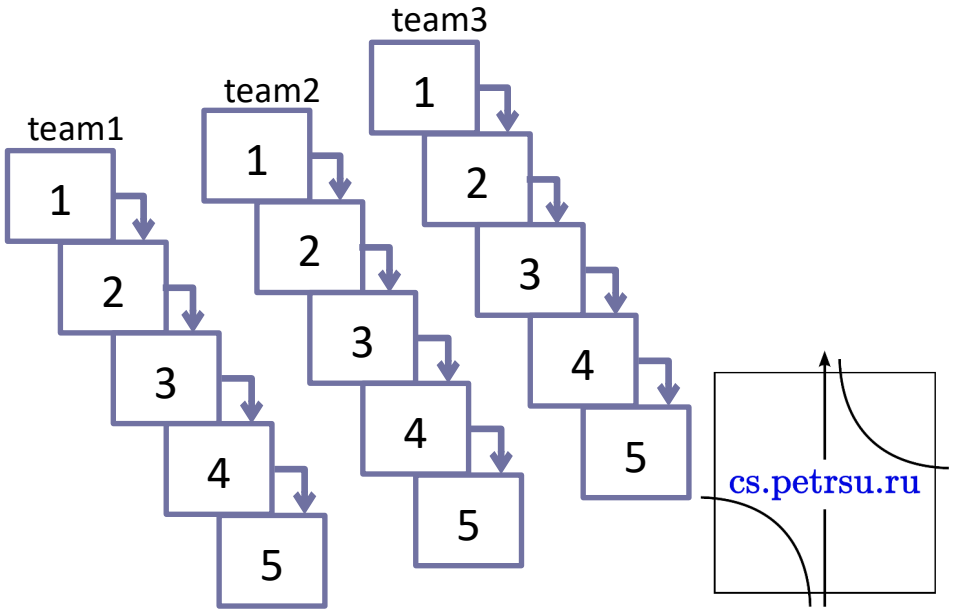
3. Моделирование обработки

   Операции над объектами данных

4. Генерация приложения

   Использование компонент

5. Тестирование



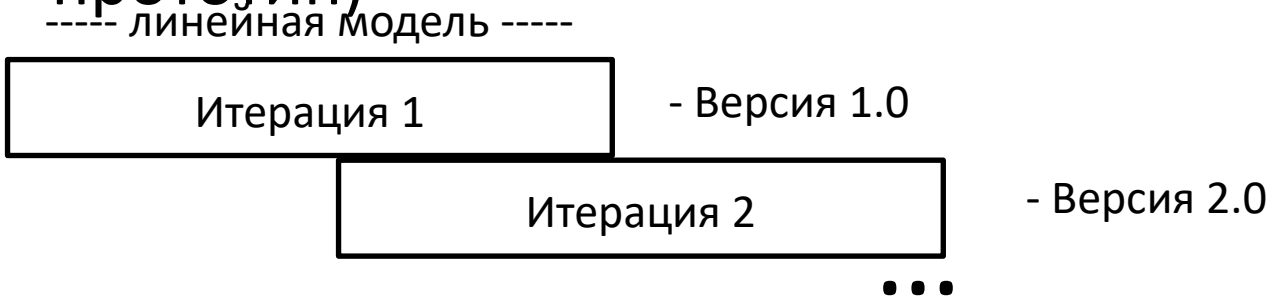
5. Эволюционная модель

На исходном этапе жизненный цикл ПО представляется весьма смутно:

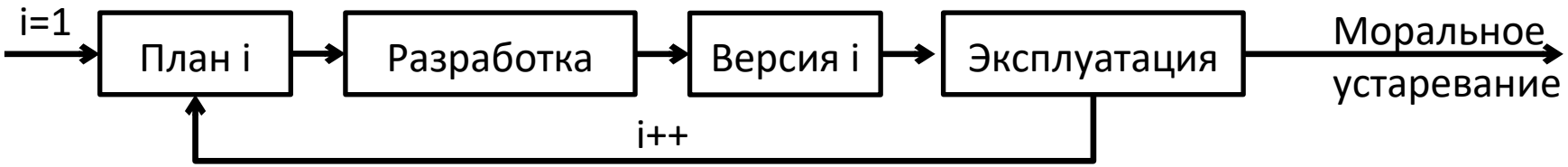
- Требования известны в общих чертах
- Круг пользователей определен не окончательно
- Может не быть явного заказчика (работа авансом – задел на будущее)
- Не определены ресурсы проекта

Т.е. ситуация сходная с прототипированием

- Идея – итеративная модель, в конце каждой итерации – некий законченных продукт (не прототип)

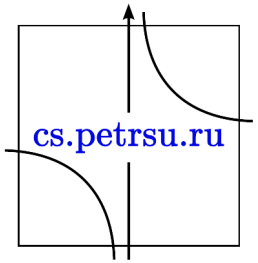


- После каждой итерации – эксплуатация полученной версии, получение новых требований и запускается следующая итерация



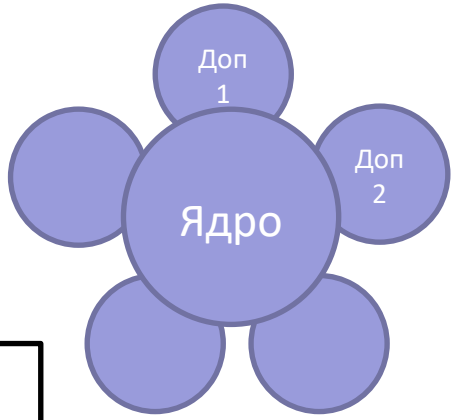
Недостатки

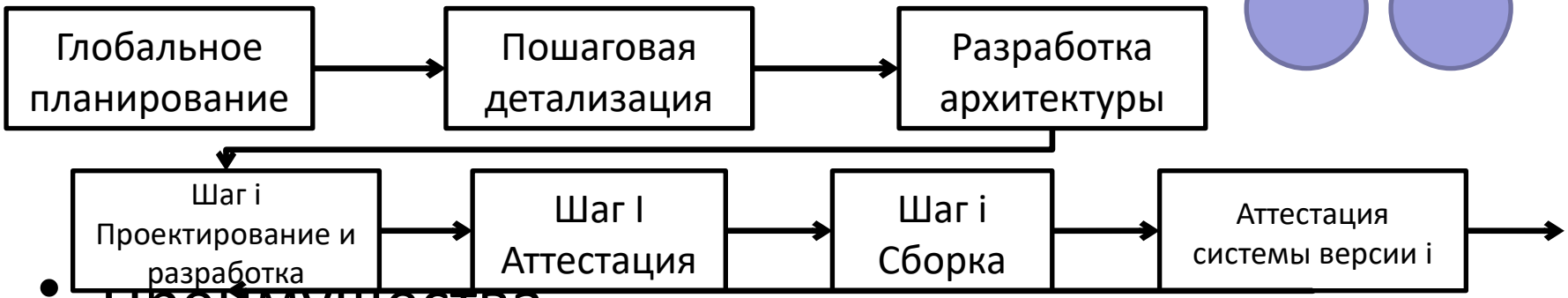
– Архитектура и внутреннее устройство ПО становится всё более громоздким и не эффективным



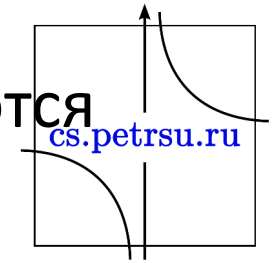
6. Пошаговая разработка (инкрементная модель)

- Требования можно разбить на:
  - Ядро
  - Доп.функции

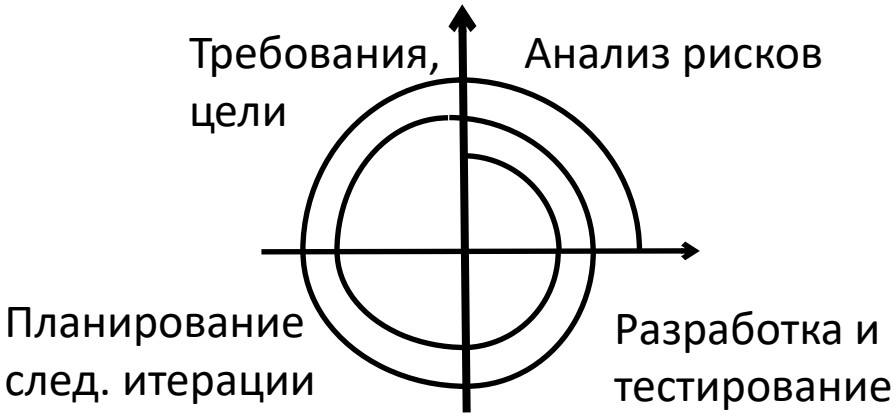




- Преимущества
  1. Заказчик все время видит нечто работающее
  2. Планируемое развитие (все определено заранее)
  3. Более важные функции – больше тестируются (начиная с первых шагов)

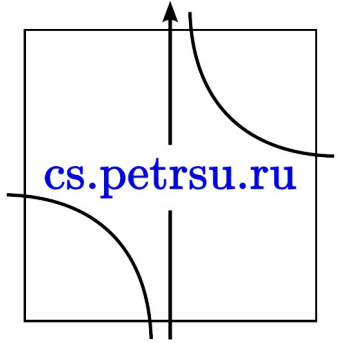


7. Спиральная модель



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

cs.petrsu.ru



Управление проектом

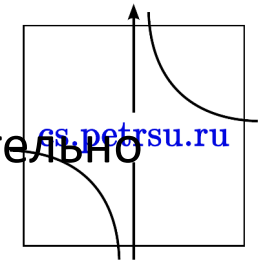
Глава №2

§0 Процессы управления

Понимание разницы между профессиональной разработкой ПО и любительским программированием.

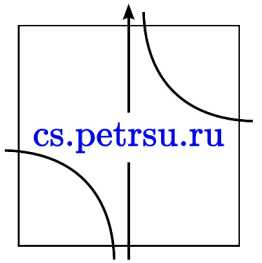
- Зоны ответственности менеджера:
  - Написание предложений по созданию ПО
  - Подбор и организация персонала
  - Планирование и график работ
  - Оценка стоимости проекта
  - Контроль хода выполнения работ
  - Написание отчетов

- Менеджер должен:
  - Гарантировать выполнение бюджетных и временных ограничений с учетом бизнес-целей организации относительно разрабатываемого ПО



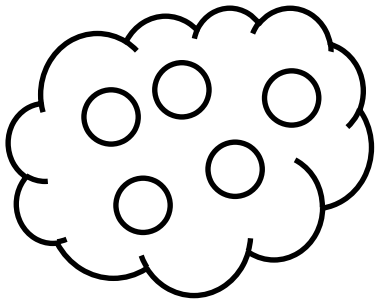
§1 Предложение по созданию ПО (обзор проекта)

- Описание проекта и разрабатываемого ПО (цели проекта и способы их достижения)
- Цель и назначение ПО
- Предметная область
- Важнейшие функции ПО
- Ограничения
- Ресурсы

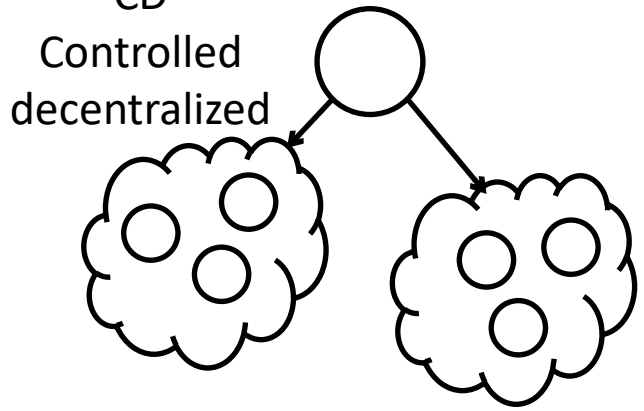


§2 Подбор персонала

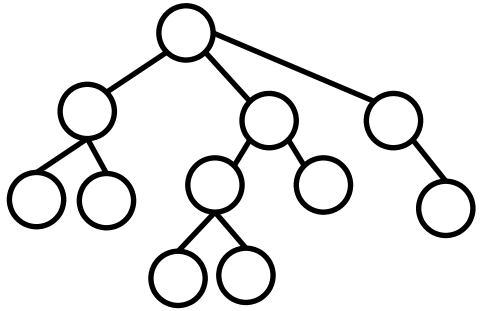
- Команда разработчиков (состав, квалификация, роли)
- Организация:



DD
Democratic
decentralized



CD
Controlled
decentralized



CC
Controlled
centralized

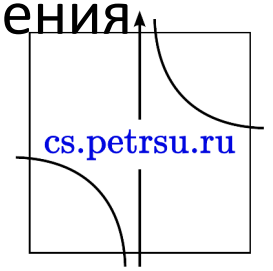
cs.petrsu.ru

§3 Планирование и график работ

1. Виды планов

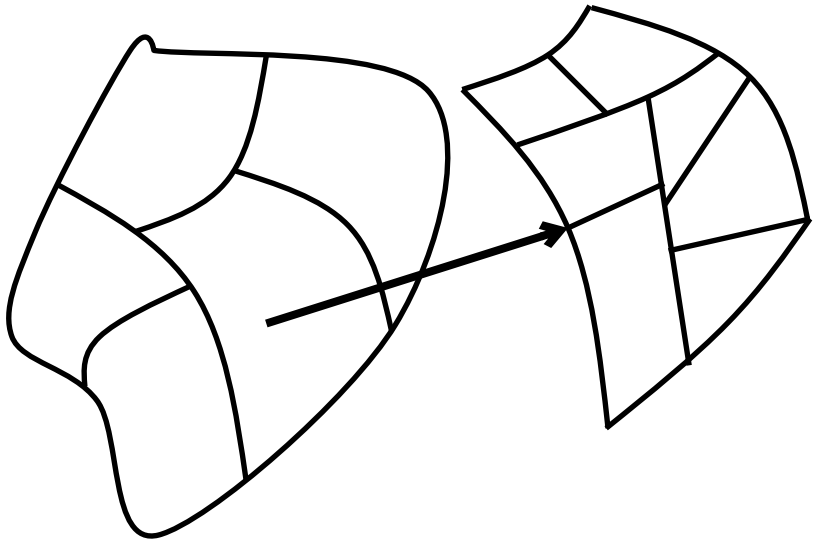
- План качества (стандарты и мероприятия по поддержке качества ПО)
- План аттестации (способы, ресурсы и перечень работ для аттестации ПО)
- План управления конфигурацией (структура и процессы управления конфигурацией)
- План сопровождения ПО (мероприятия по сопровождению ПО в процессе его эксплуатации)
- План управления персоналом (мероприятия по взаимодействию разработчиков и повышению квалификации)

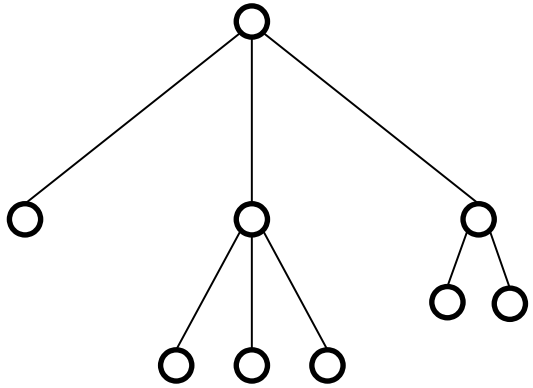
- План регулярно пересматривается в процессе управления проектом
- Планирование – итеративный процесс (новая информация, возникновение проблем, …)



2. Декомпозиция задач

Обход в ширину, а не в глубину





3. Контрольные отметки этапов работ

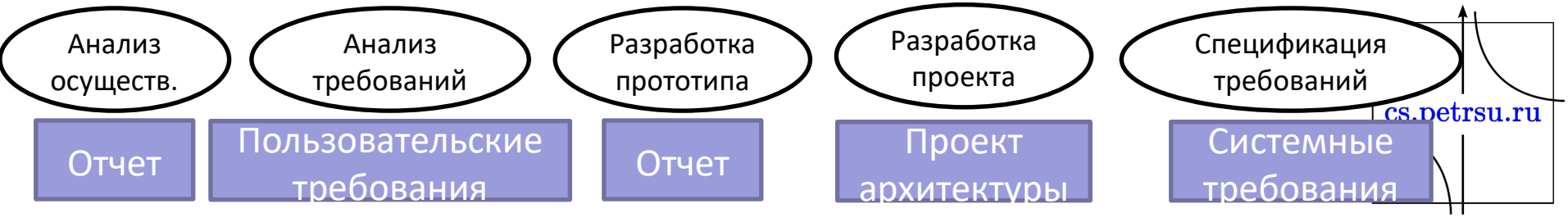
Для управления нужно регулярное получение информации

Управленческая информация – в виде документов, отображающих выполнение очередного этапа

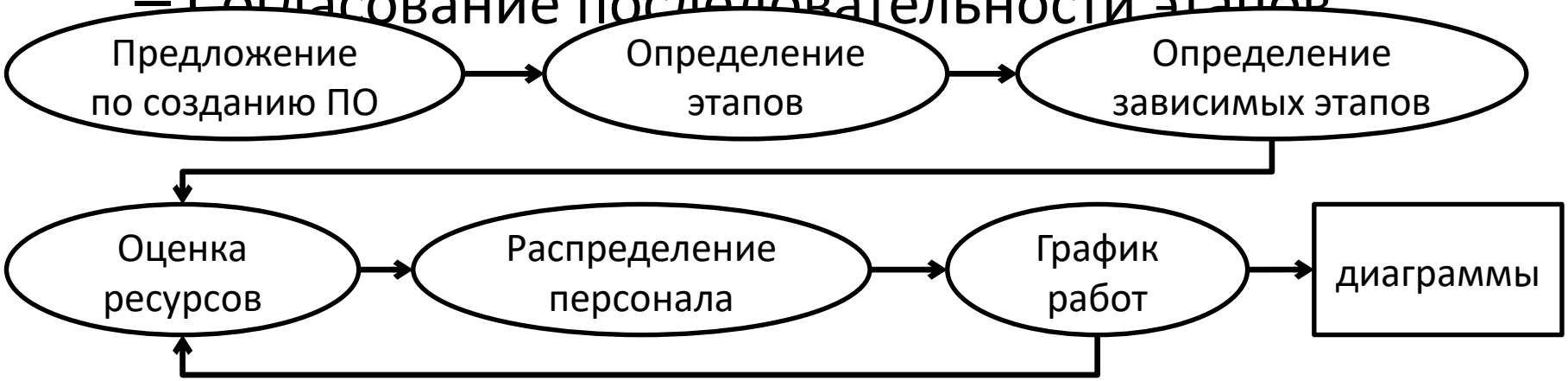
cs.petrsu.ru

- Степень готовности продукта
- Оценка произведенных затрат
- Коррекция графика работ

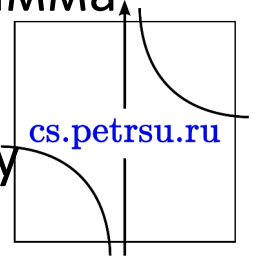
- Контрольные отметки – вехи, отмечающие окончание этапа работ
- Отчет для руководства проекта
- Контрольные проектные элементы – документация, прототип, подсистем и т.д.



- 4. График работ
  - Оценка длительности проекта и его этапов
  - Ресурсы для реализации отдельных этапов
  - Согласование последовательности этапов

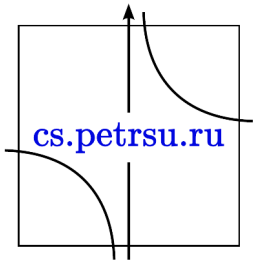


- 5. Временные и сетевые диаграммы
  - Временная диаграмма – показывает время начала и окончания каждого этапа и его длительность. Диаграмма Гантта
  - Сетевая диаграмма – отображает зависимость между различными этапами. Диаграмма Перта.



§4 Контроль хода выполнения работ (прогресс разработки)

- Объекты контроля
  - Регулярность работ
  - Обеспечение качества
- Способы
  1. Собрание команды разработчиков
     - Обсуждение
     - Протоколы собраний
  2. Индивидуальные отчеты разработчиков
     - Порученные задачи
     - Состояние решаемых задач
  3. Отчет о текущем состоянии проекта
     - Соответствие плану
     - Результаты
     - Потраченные и необходимые ресурсы
     - Прогнозы



§5 Другие зоны ответственности менеджера

1. Оценка затрат

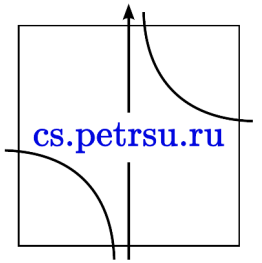
- Людские ресурсы (человеко-часы, человеко-месяцы, …)
- Продолжительность (календарные дни, …)
- Стоимость

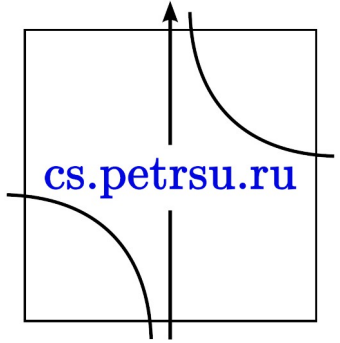
Основа – прошлый опыт

- Количественные измерения: метрики(показатели), напр. LOC
  - Стоимость разработки и реализации функции
  - Анализ чувствительности проекта

2. Написание отчетов; документация

- Постепенное наращивание (детализация)
- Визуальный язык (диаграммы, схемы)
- Структурирование
- Анализ альтернатив и обоснование выбора
- Формы отчетности (устные, письменные, email, web, …)





Требования к ПО

Глава №3

§1 Понятие “требования”

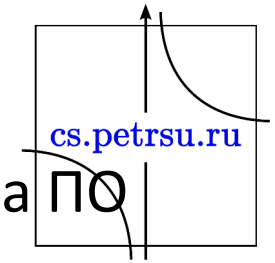
При создании ПО решается множество проблем.

Природа этих проблем не всегда ясна, напр. трудно четко представить те действия, которые должна выполнять система.

Опр.: Описание функциональных возможностей и ограничений, накладываемых на ПО, называется **требованиями**.

Процесс формирования, анализа, документирования и проверка этих функциональных возможностей и ограничений – **разработка требований**.

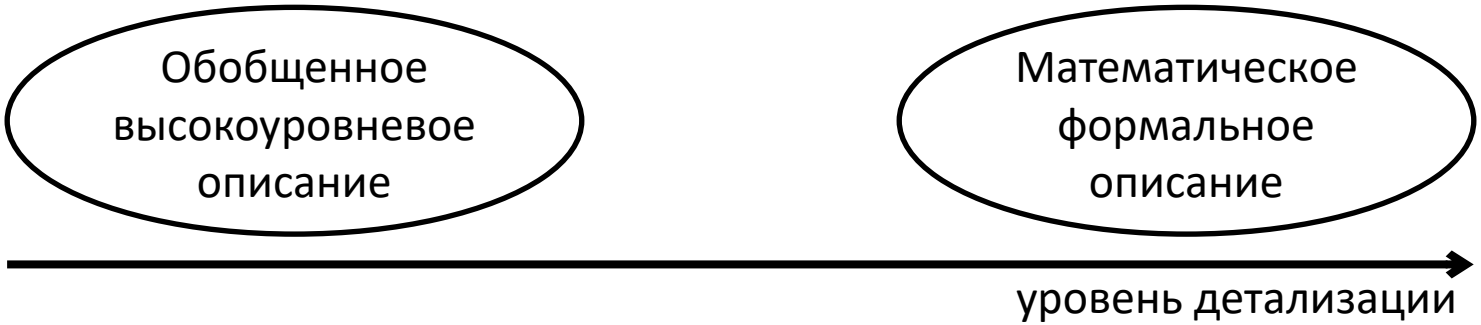
Requirements engineering – первый этап производства ПО



ЧТО - требования это ответ на это вопрос

Уровни детализации

- Нельзя сразу полностью решить сложную задачу
- Постепенное итеративное наращивание результатов (анализ проблемы в ширину)



- Разделение требований на уровни

cs.petrsu.ru

I. Пользовательские требования

- Описание на естественном языке + поясняющие диаграммы
- Понятно заказчику и пользователю

II. Системные требования

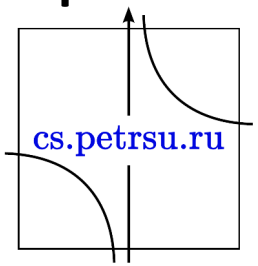
- Детализированное описание функций и ограничений (функциональная спецификация)
- Формализованные языки (моделирование: UML, математические модели)
- Понятно экспертам заказчика и разработчикам

III. Спецификация требований (проектная системная спецификация или ТЗ)

- Первичный список требований (заказчик сам, интервьюирование, анкетирование, наблюдение, прототипирование)
- Модели требований
- Высокоуровневая архитектура системы
- Критерии аттестации системы

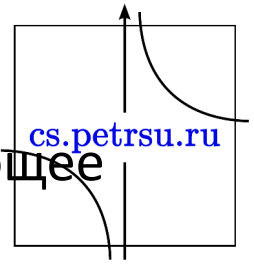
I

II



Примеры требований

- Пользовательское требование
  1. ПО должно предоставить средство доступа к внешним файлам, созданным в других программах
- Системные требования
  1. Пользователь должен иметь возможность определять тип внешних файлов
  2. В зависимости от типа внешнего файла должно быть соответствующее средство, применимое к этому типу
  3. Внешний файл любого типа должен быть представлен соответствующей пиктограммой
  4. При выборе пользователем пиктограммы, представляющей конкретный внешний файл, к этому файлу должно быть применено средство, ассоциирующее внешними файлами данного типа.

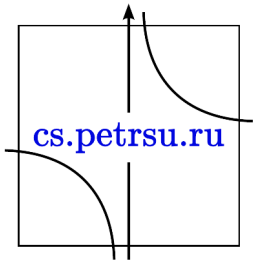


§2 Функциональные и не функциональные требования

1. Функциональные требования

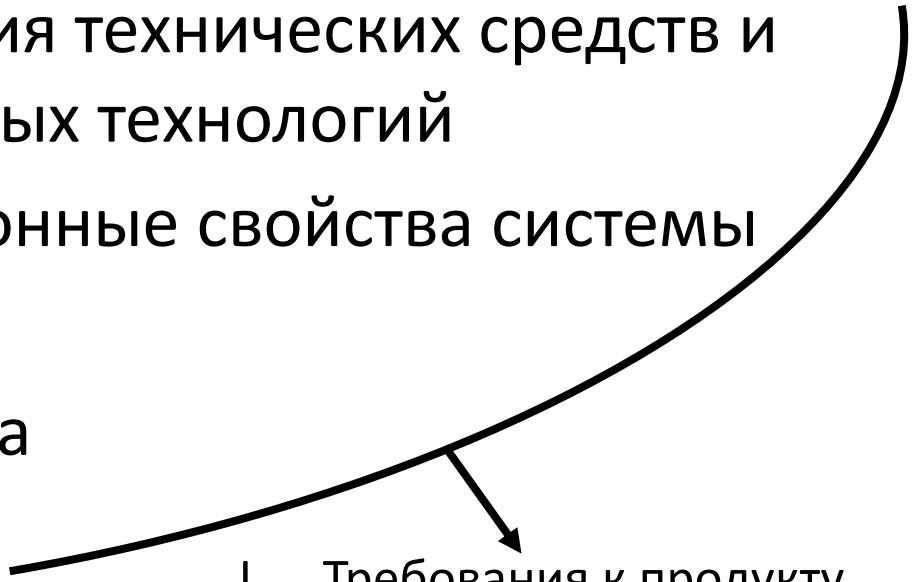
- Перечень функций, услуг, сервисов, которые должна выполнять система
- Должно быть так же указано, как система реагирует
  - на те или иные входные данные,
  - как ведет себя в определенных ситуациях,
  - Иногда следует указать, что система не должна делать

- Шаблон описания:
  - Пользователь должен иметь возможность …
  - Система должна предоставлять …



2. Нефункциональные требования (атрибуты системы)

- Ограничения на выполняемые функции
- Ограничения предметной области
- Ограничения рамок системы
- Ограничения технических средств и программных технологий
- Интеграционные свойства системы
- Интерфейс
- Архитектура



I. Требования к продукту
II. Организационные требования
III. Внешние требования

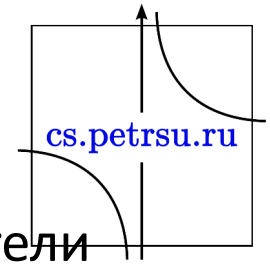
cs.petrsu.ru

Примеры

- Безопасность
- Производительность
- Структуры данных
- Техника безопасности
- Законодательство
- Организационные требования
- Предметная область
- Глоссарий терминов
- Рамки системы
- Архитектура

Сложность проверки нефункциональных требований

- Нечетко сформулированы
- Затруднительно использовать количественные показатели



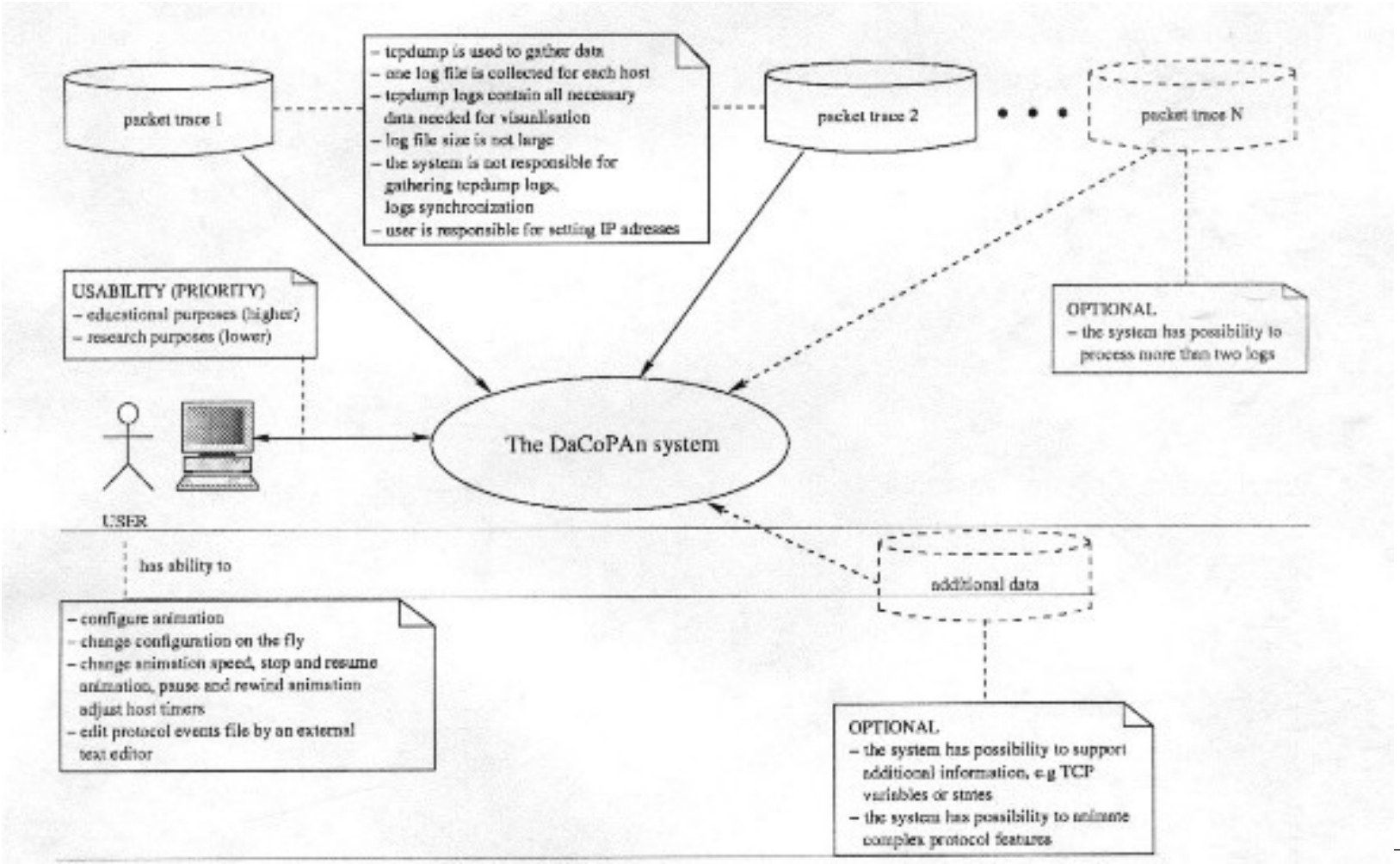


Figure 1: System scope model

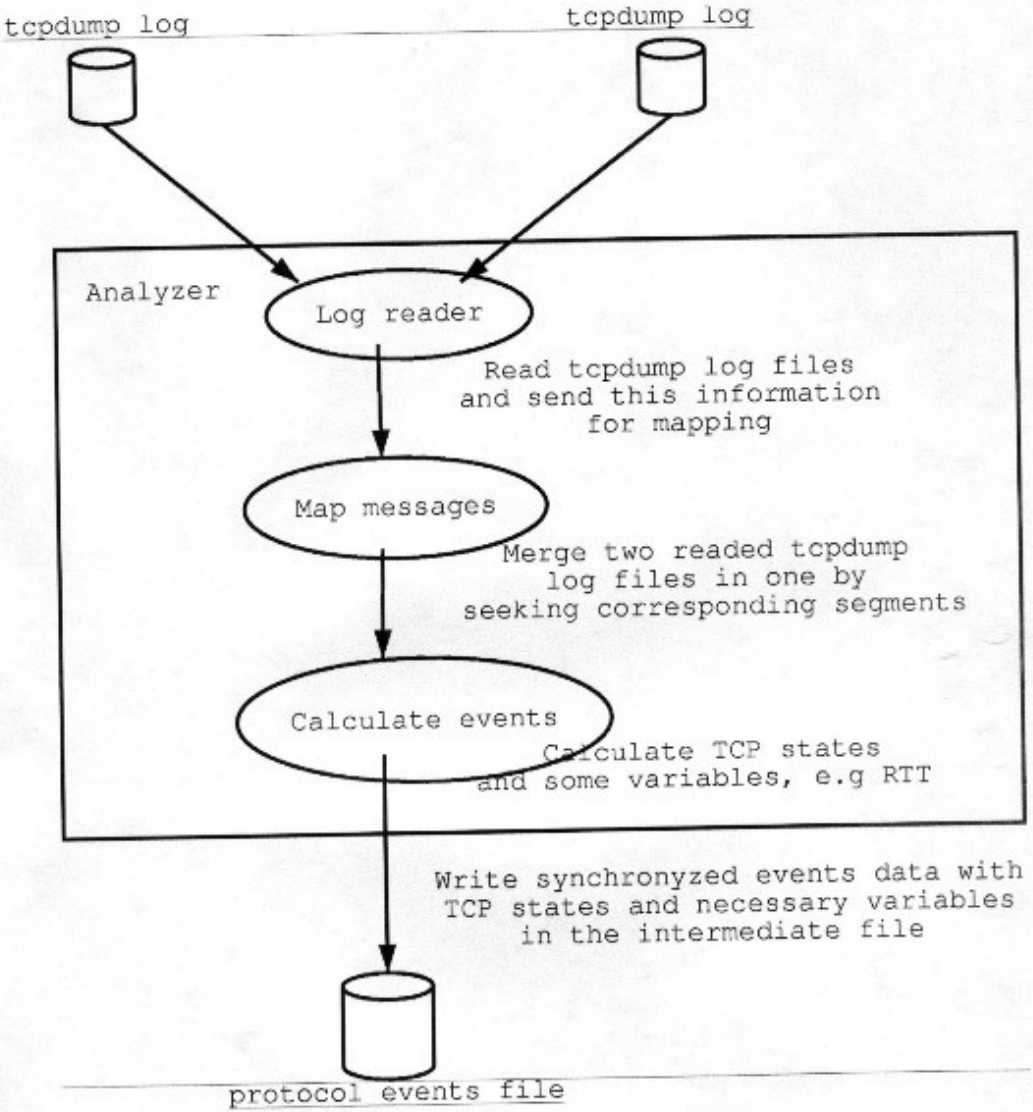
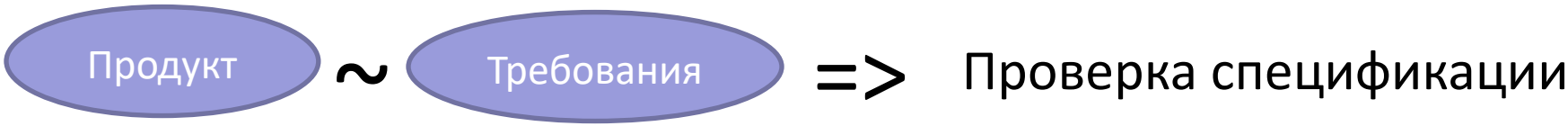


Figure 3: Functions of Analyzer

Качественные показатели для нефункциональных требований

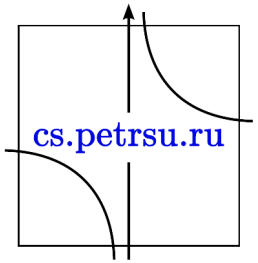
Показатель	Единицы измерения
Скорость	- количество выполняемых транзакций в сек. - время реакции на действие пользователя - время обновления экрана
Размер	- в Кб - Количество модулей
Простота эксплуатации	- Время обучения персонала - Количество статей в справочной системе
Надежность	- Среднее время безошибочной работы - Вероятность выхода системы из строя
Устойчивость к сбоям	- Время восстановления системы после сбоя - Процент событий приводящих к сбоям - Вероятность порчи данных при сбоях
Переносимость	- % машинно-зависимых операторов - Количество машинно-зависимых подсистем

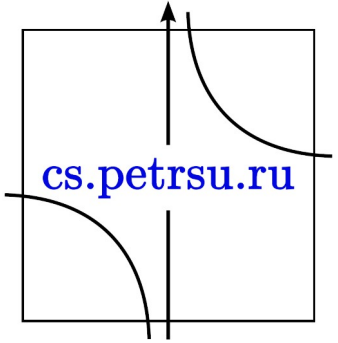
§3 Аттестация требований



- Полнота (все требуемые функции)
- Непротиворечивость (отсутствие несовместимых и взаимоисключающих функций)
- Отсутствие избыточности
- Приоритеты и риски
- Критерии аттестации

} Утверждено заказчиком И разработчиком

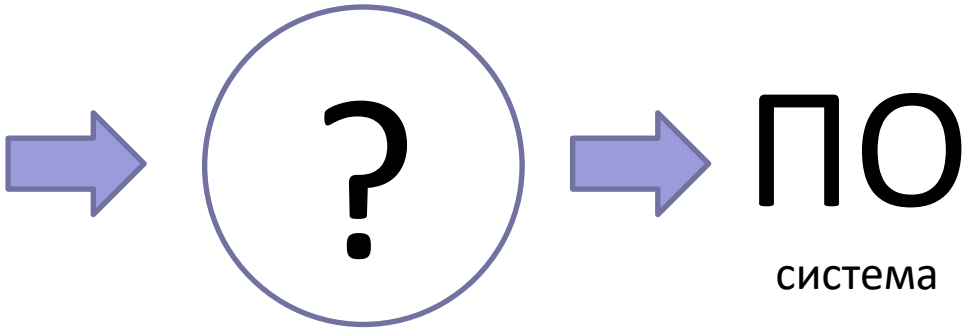




Моделирование

Глава №4

- Требования
- Информация о природе объектов
- Внешнее окружение



- Текстовое представление
- Моделирование
- Кодирование

Процесс моделирования состоит из:

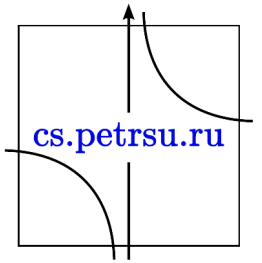
- Парадигмы(языка)
- Инструментального средства

§1 Модели и анализ

Модель – это упрощенное представление чего-либо (в некоторой среде).

Она отражает только важные с определенной точки зрения черты моделируемой сущности, а остальные игнорирует или передает в упрощенном виде.

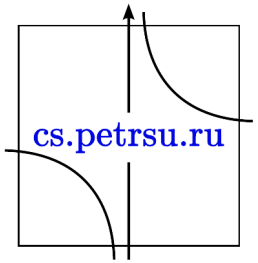
- аналитические модели
- имитационные модели



Цели моделирования:

- Описание сущностей
- Описание процессов
- Описание связей
- Детализация и декомпозиция

- ✓ Понять как устроен конкретный объект, исследовать его структуру
- ✓ Научится управлять объектом и/или процессом
- ✓ Создание объектов с заданными свойствами оригинала
- ✓ Эффективность управления объектом и/или процессом

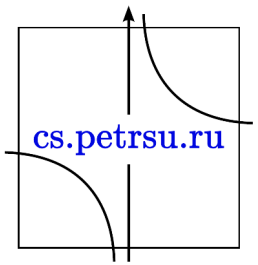


Цели моделирования:

1. Точное описание требований к системе и знаний о предметной области
   - Чтобы вовлеченные в проект стороны могли их понять и прийти к общему соглашению
2. Структура системы
   - Исследование возможных архитектурных и проектных решений до написания кода
3. Отображение проектных решений разного рода
   - Поведение системы
   - Структура данных
   - Состояния системы
   - Потоки данных

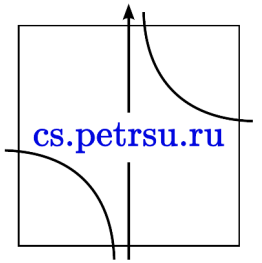
   } Разносторонний взгляд на систему
4. Создание промежуточных продуктов
   - Смета расходов
   - Проектирование форм пользовательского интерфейса
   - Сценарии использования системы (руководство пользователя)
5. Комплексные взгляд на систему с различной детализацией

   → Выбор наиболее эффективных проектных решений
6. Борьба со сложностью ПО



Принципы моделирования

1. Разбиение на уровни (постепенная детализация, исследование результатов)
2. Визуальное моделирование (диаграммы + текст)
3. Абстракция, а не полная детализация
4. Спецификация, а не реализация



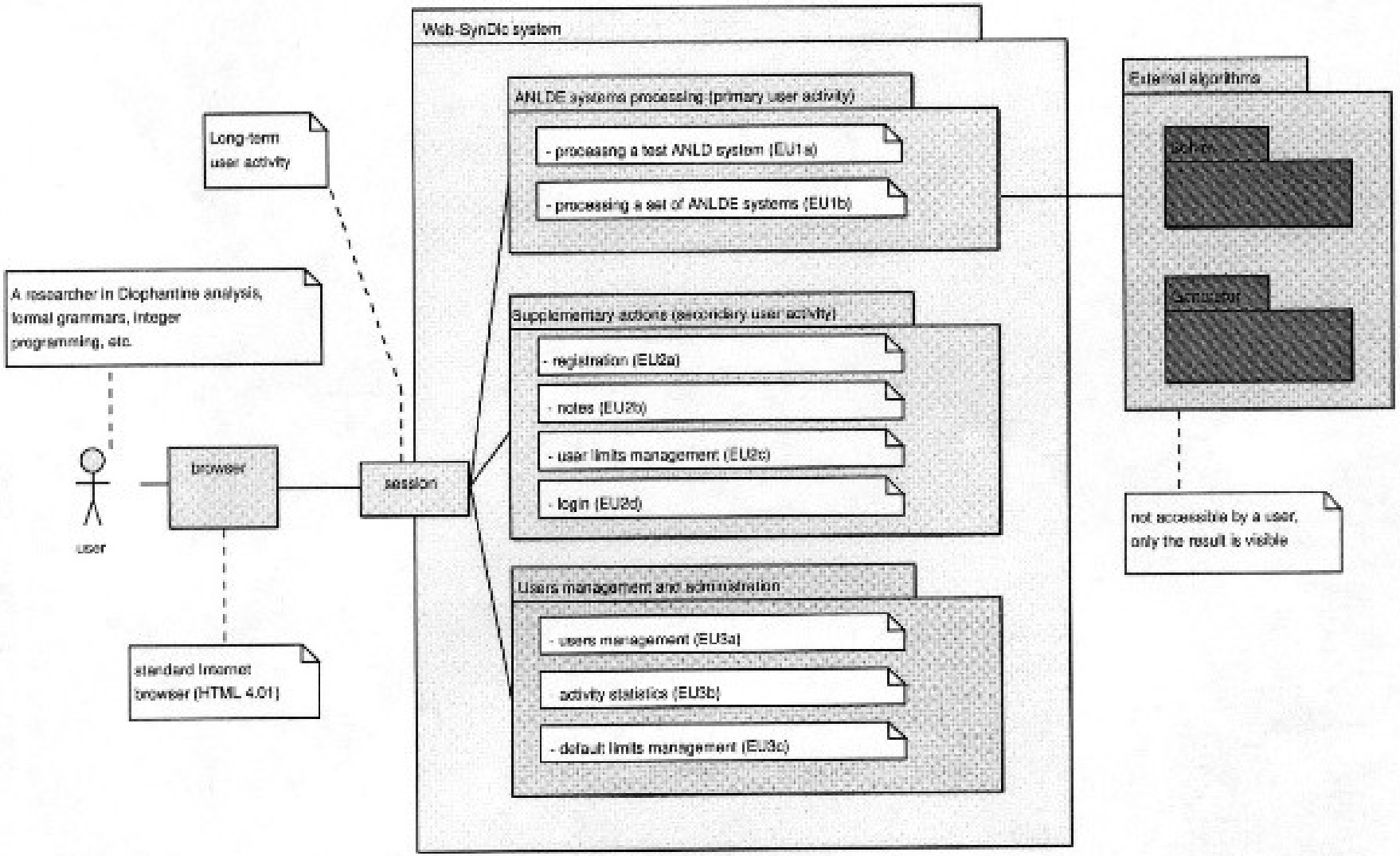


Figure 2: Structure of the problem domain

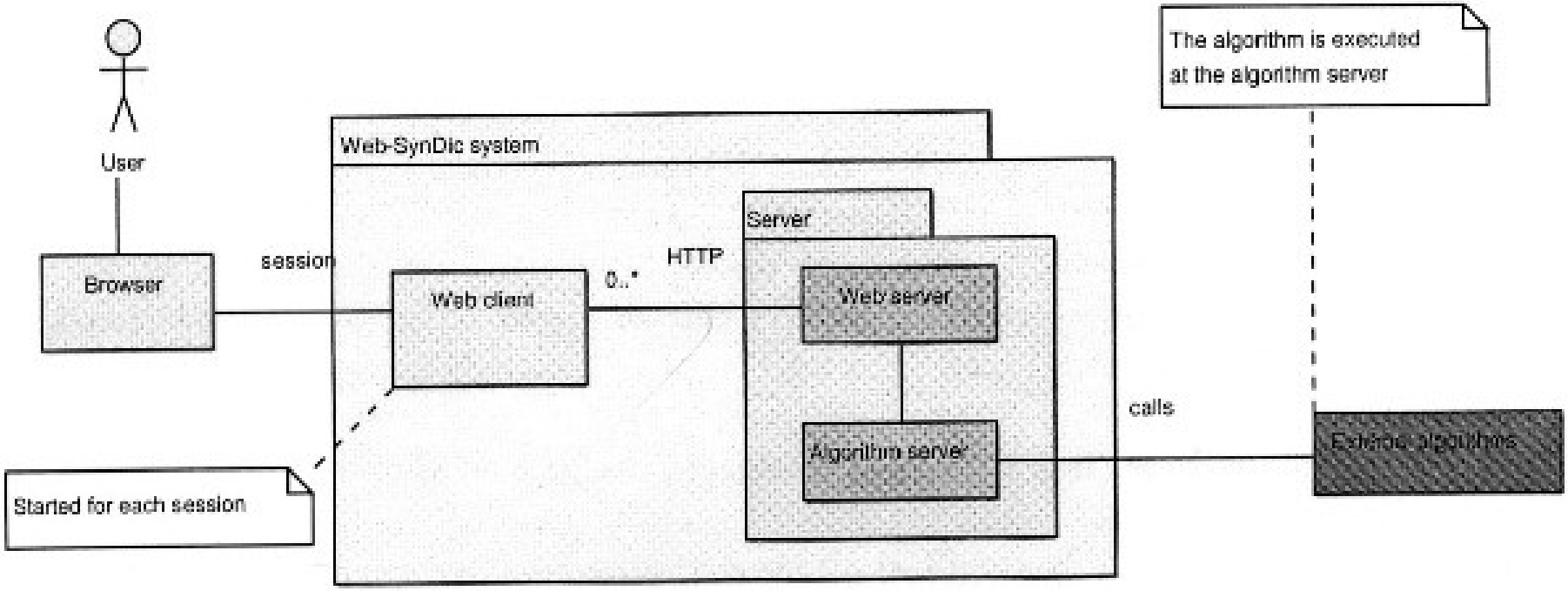


Figure 3: The composite model for high-level objects of the problem domain

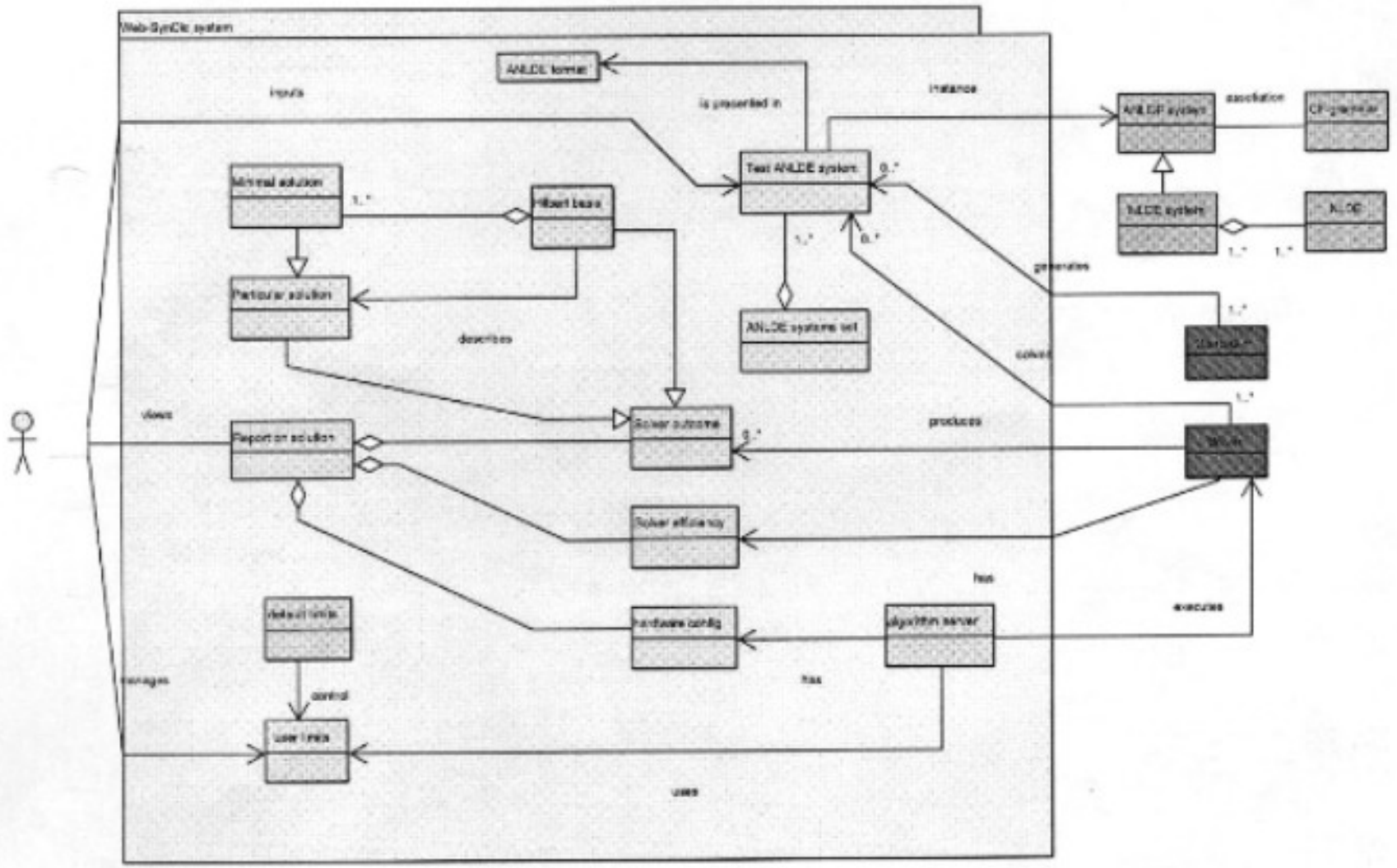


Figure 4: Composition model for ANLDE system processing

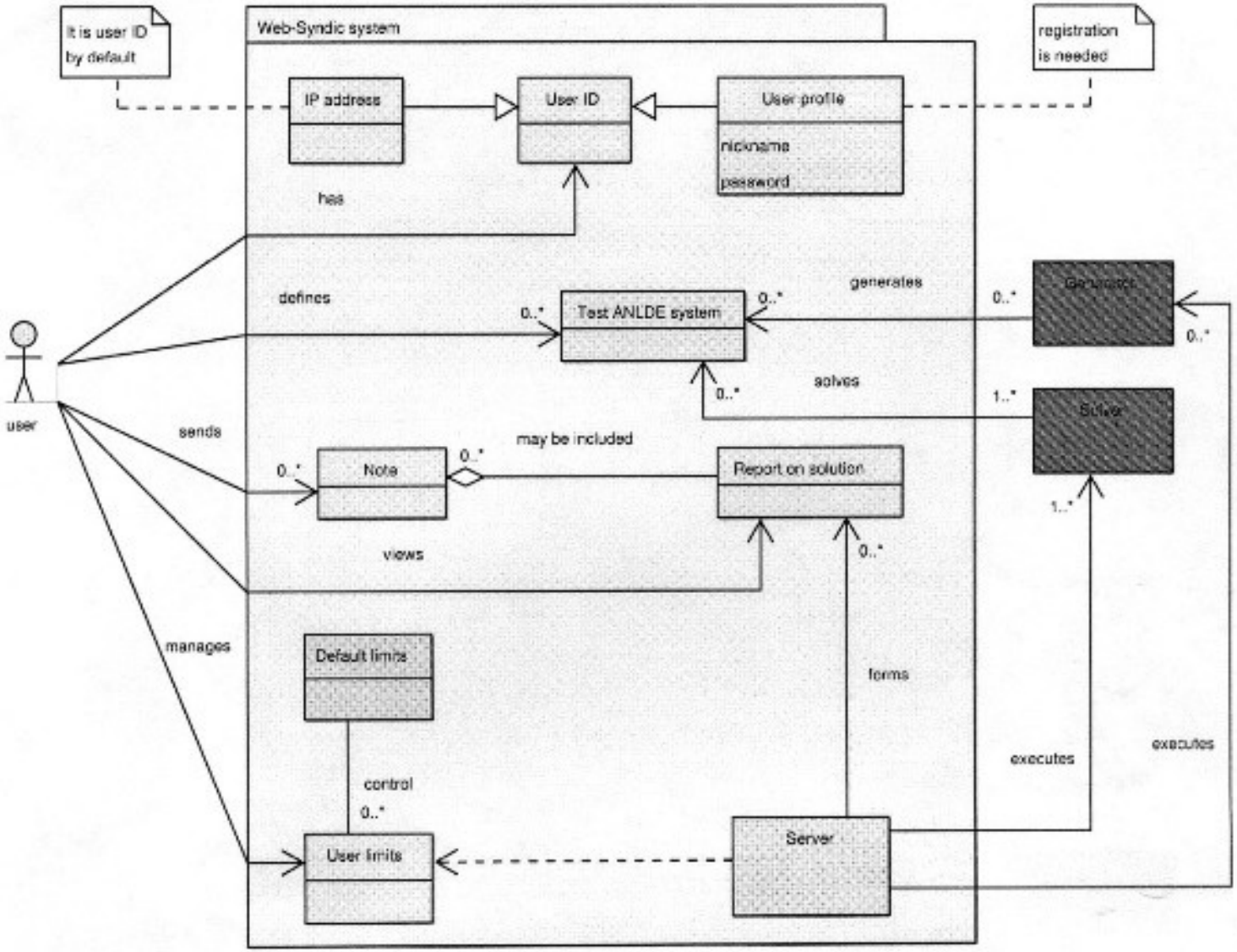


Figure 5: Composition model for supplementary user actions

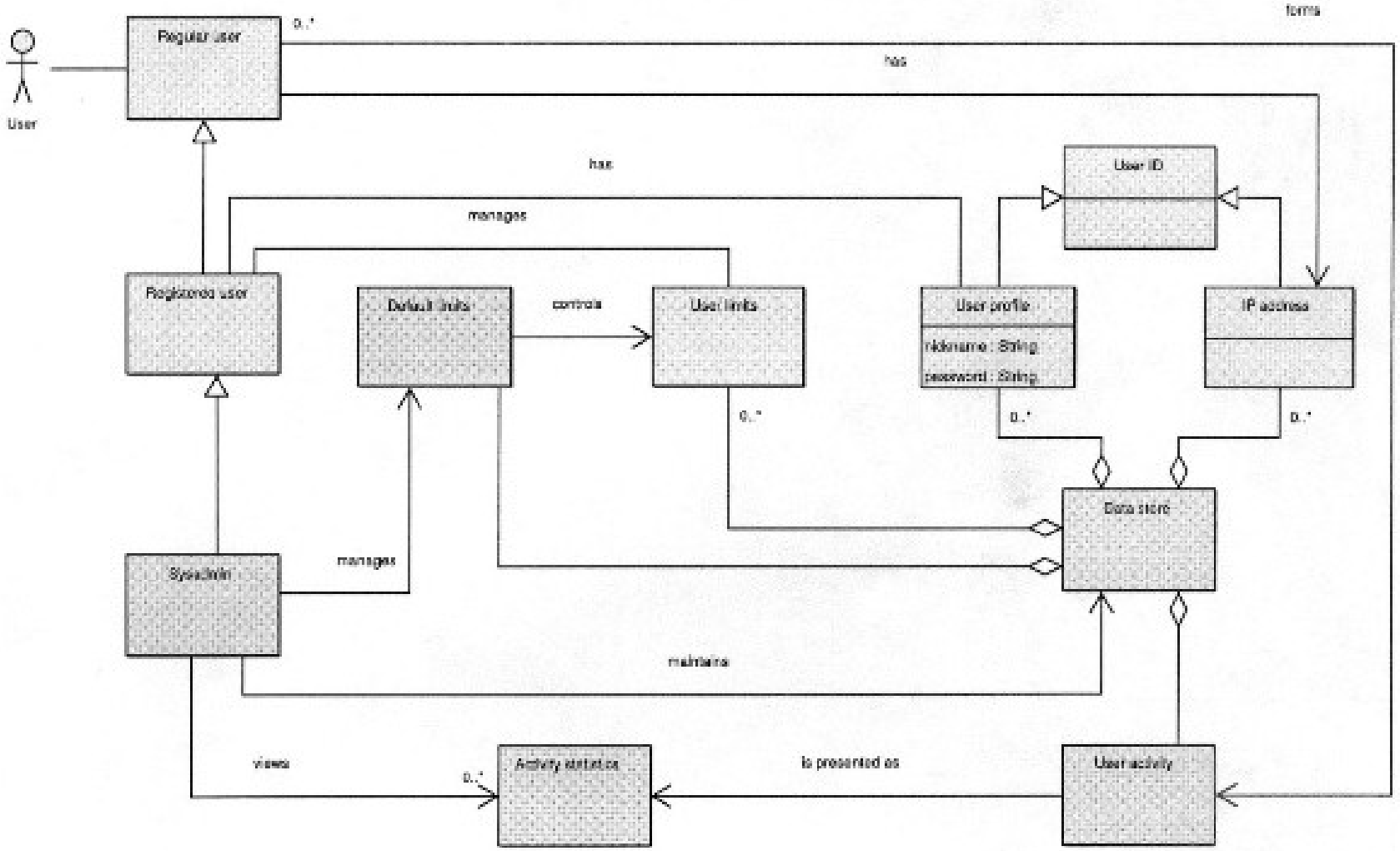
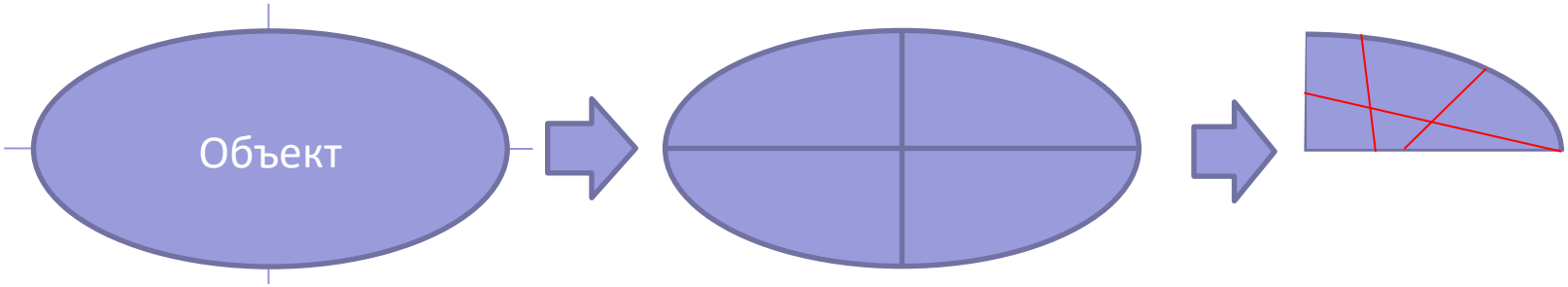


Figure 6: Composition model for user management and administration

Классический подход

- Декомпозиция объекта



Связи с внешним миром (система, функции)

Подсистемы процессы

Модели действия

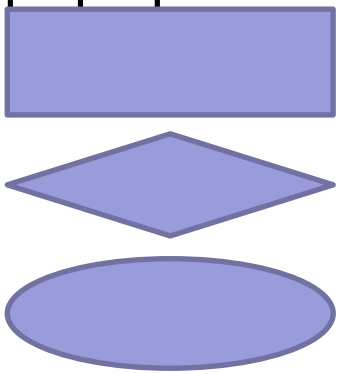
- Использование шаблонов
- Отбрасывание мелких деталей

§2 Диаграмма сущность-связь

ERD – Entity Relationship Diagram

Питер Чен в 1976, он же автор графической нотации

- Сущность / Объект
- Связь / Отношение
- Атрибут объекта
  - Особенности:
    - Концептуальные схемы предметной области
    - Моделирование системного окружения
    - Представление структурных данных
    - Моделирование БД

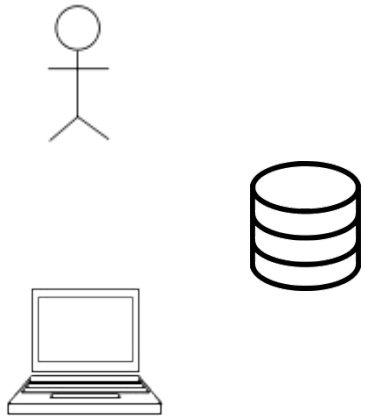


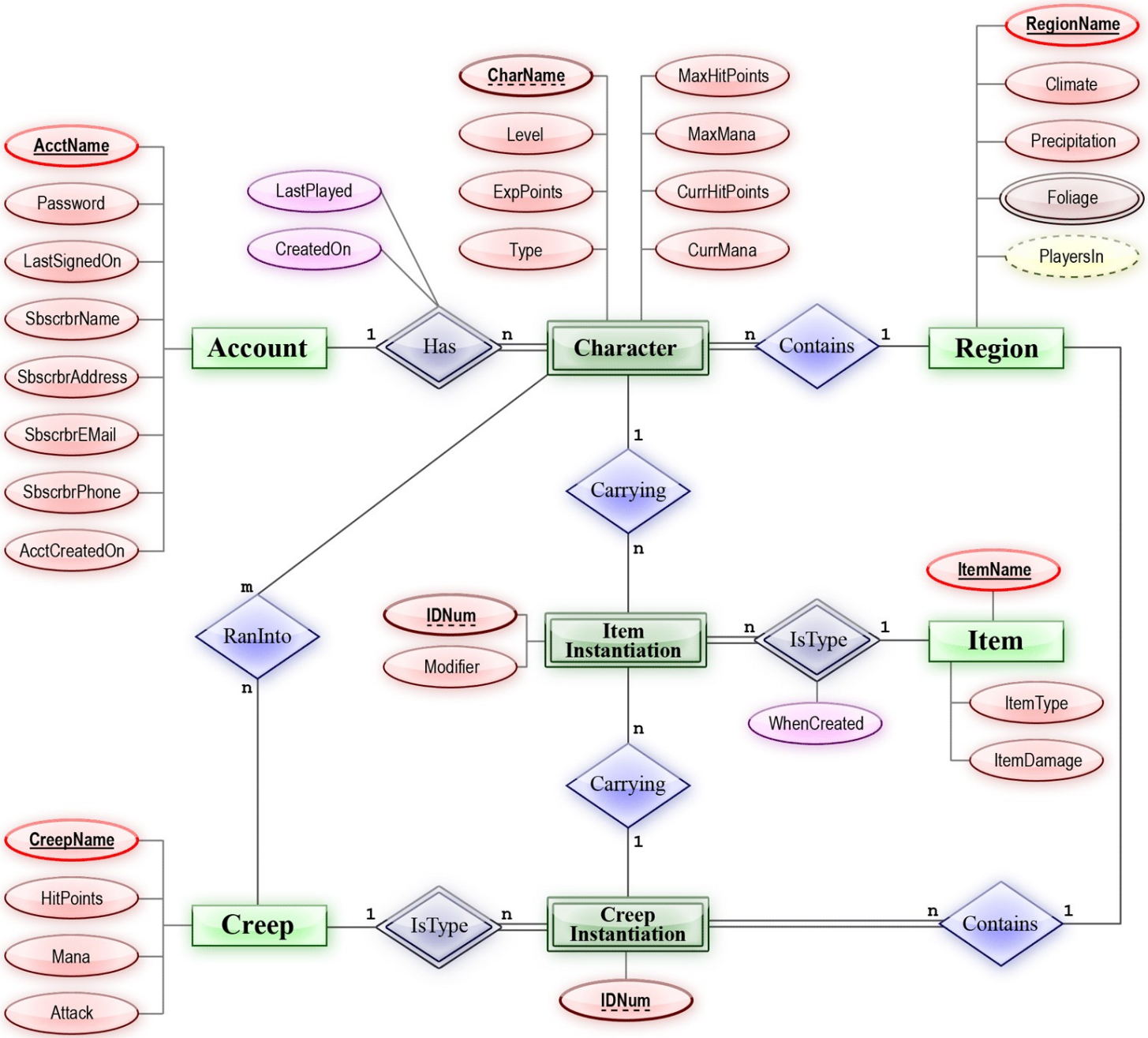
Кардинальность
Модальность
Направление

cs.petrsu.ru

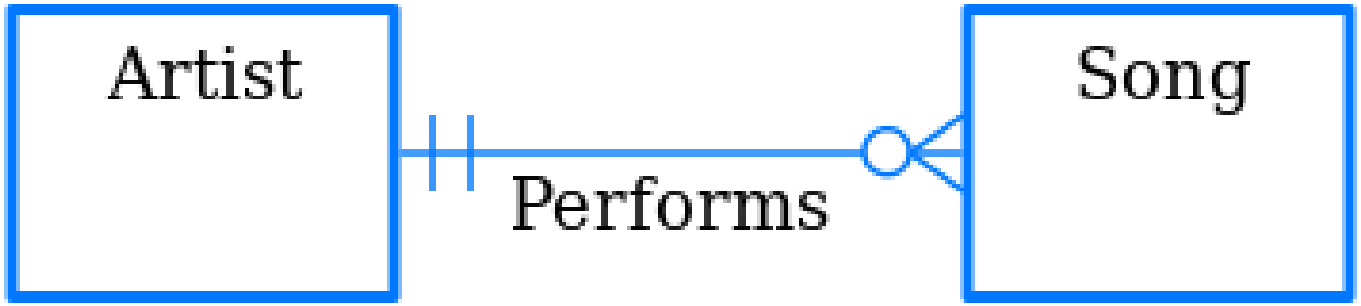
В некоторых случаях возникают специальные обозначения:

- Инициатор внешних событий
- Хранилище данных
- Терминал
- Более слабая (опциональная) связь




RegionName
Climate
Precipitation
Foliage
PlayersIn
CharName
Level
ExpPoints
Type
MaxHitPoints
MaxMana
CurrHitPoints
CurrMana
AcctName
Password
LastSignedOn
SbscrbrName
SbscrbrAddress
SbscrbrEMail
SbscrbrPhone
AcctCreatedOn
LastPlayed
CreatedOn
Account
1
Has
n
Character
n
Contains
1
Region
1
Carrying
n
m
RanInto
n
IDNum
Modifier
Item Instantiation
n
IsType
1
Item
ItemName
ItemType
ItemDamage
WhenCreated
n
Carrying
1
CreepName
HitPoints
Mana
Attack
Creep
1
IsType
n
Creep Instantiation
n
Contains
1
IDNum

- Нотация Crow’s Foot («воронья лапка»)

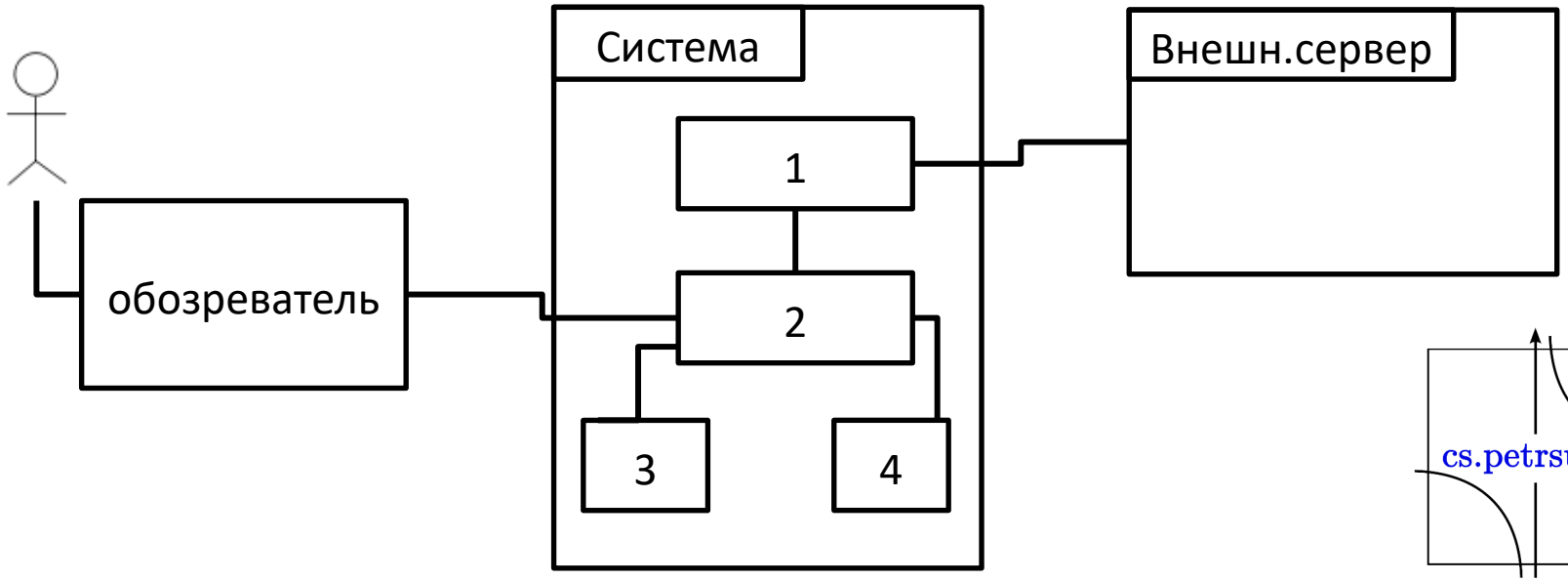


§3 Глоссарий и предметная область

- Список терминов предметной области и/или проекта с определениями.

  Термины: объекты (сущности), связи/функции (глаголы).

- Структуризация глоссария:



- Высокоуровневые объекты:
  - Пользователь, обозреватель, 1, 2, 3, 4, внешний сервер.
- Каждый высокоуровневый объект состоит из более мелких – отдельная ERD и часть глоссария

Назначение глоссария:

Сформировать единое понимание предметной области, системы, системного окружения и рамок системы.

§4 Модель потоков данных (Data Flow Diagram) DFD

- Внешние сущности

  Элемент системы или внешний источник

  Аппаратное средство, пользователь, программа

- Системы / процессы

  Или трансформация которая применяется к данным и изменяет(преобразует) их

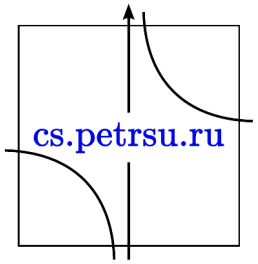
- Накопители данных

- Потоки данных

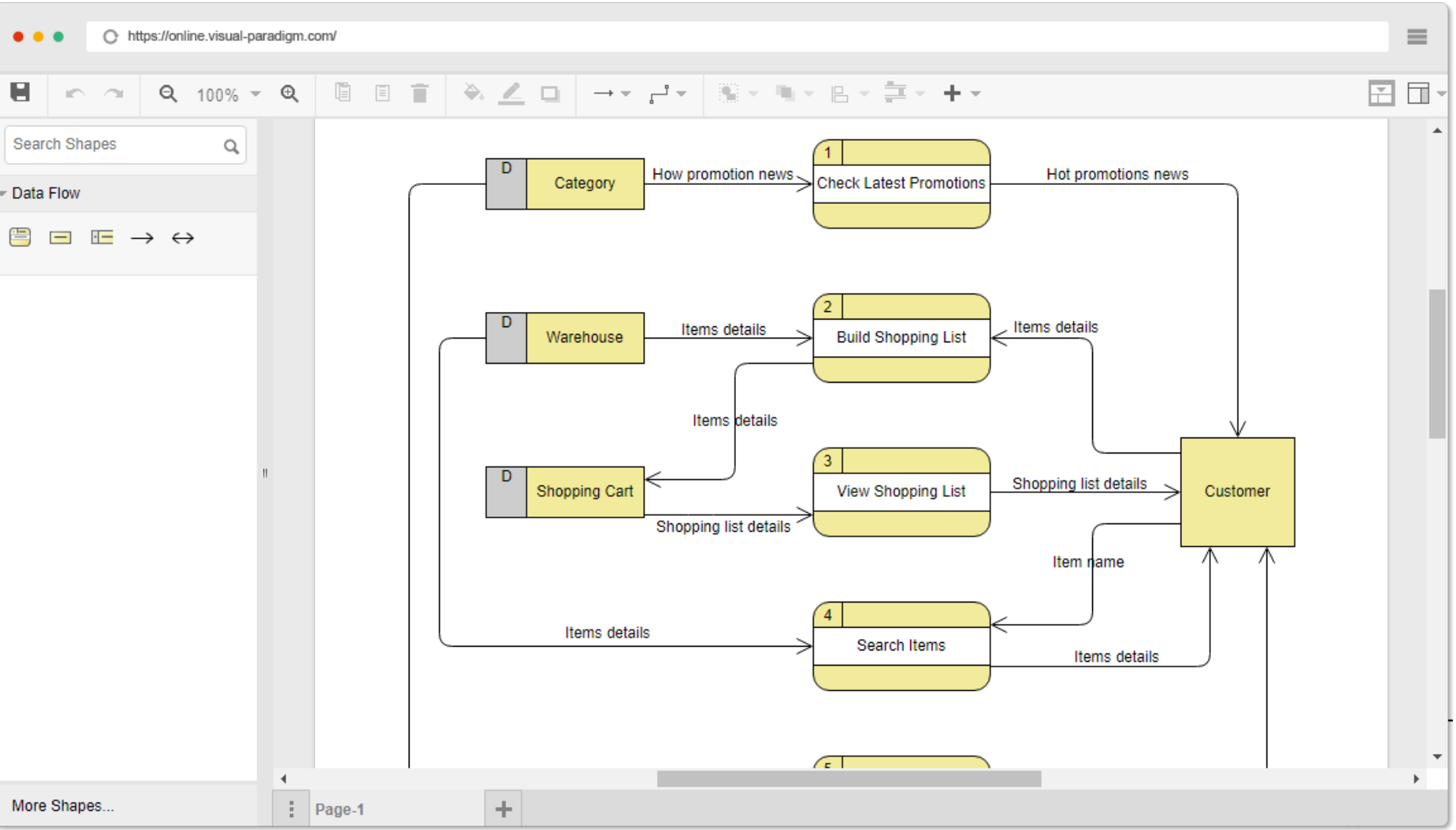
  Или объект данных

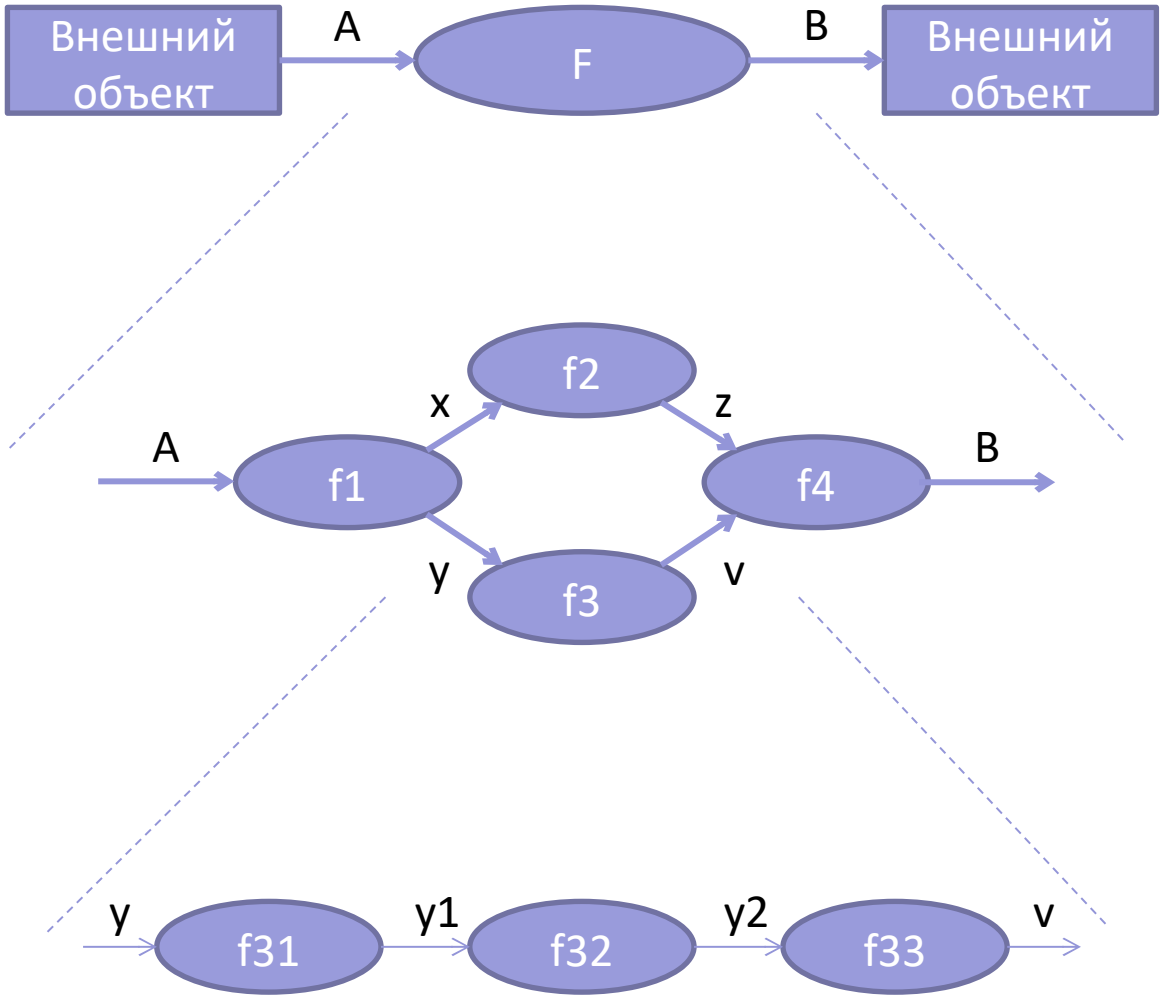
нотация Гейна-Сарсона

* Используется на любом уровне абстракции для изображения информационных потоков и их преобразований



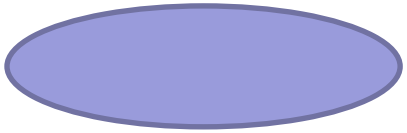
Пример DFD

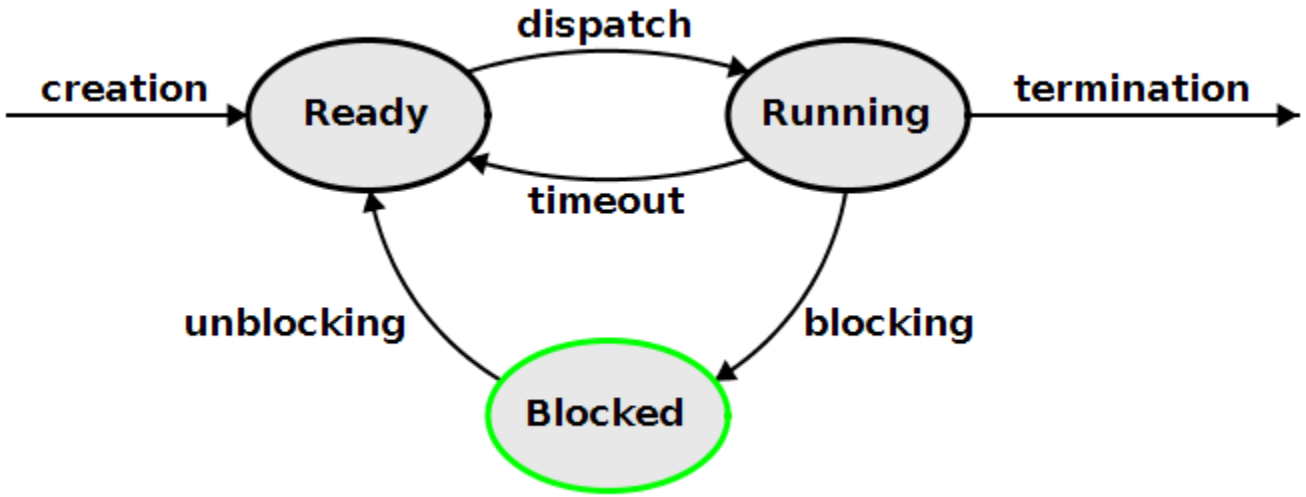

https://online.visual-paradigm.com/
100%
Search Shapes
Data Flow
More Shapes...
Page-1
D
Category
How promotion news
1
Check Latest Promotions
Hot promotions news
D
Warehouse
Items details
2
Build Shopping List
Items details
Items details
D
Shopping Cart
3
View Shopping List
Shopping list details
Customer
Shopping list details
Item name
4
Search Items
Items details
Items details


Внешний объект
A
F
B
Внешний объект
1 уровень
f2
x
z
A
f1
f4
B
2 уровень
y
v
f3
y
y1
y2
v
f31
f32
f33
3 уровень

§5 Диаграммы переходов и состояний

(State Definition Diagram) STD

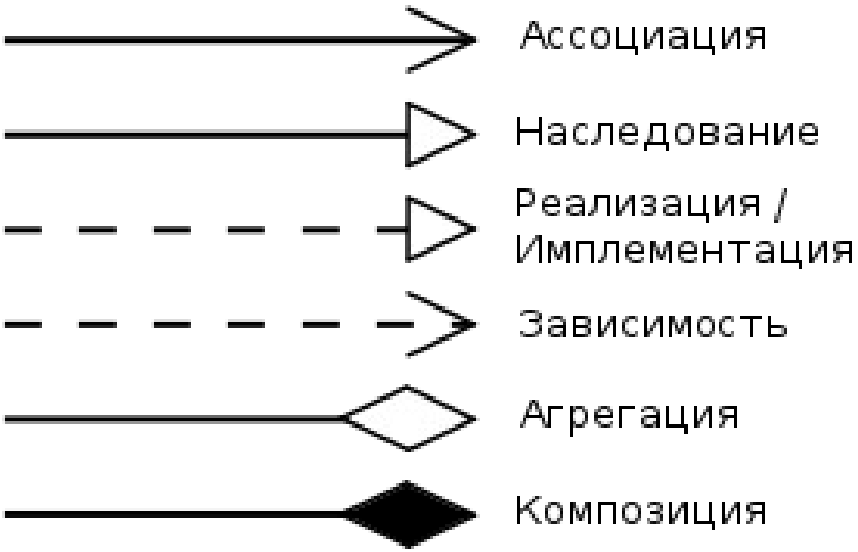
- Состояния 
- Переходы 



* Она же - модель конечных автоматов

§6 Объектные модели
(Диаграмма классов)

- Уровень анализа классов (набросок контуров и концепция предметной области)
- Уровне проектирования классов (основные проектные решения)
- Уровне реализации классов (дорабатывается до удобной реализации на языке программирования)

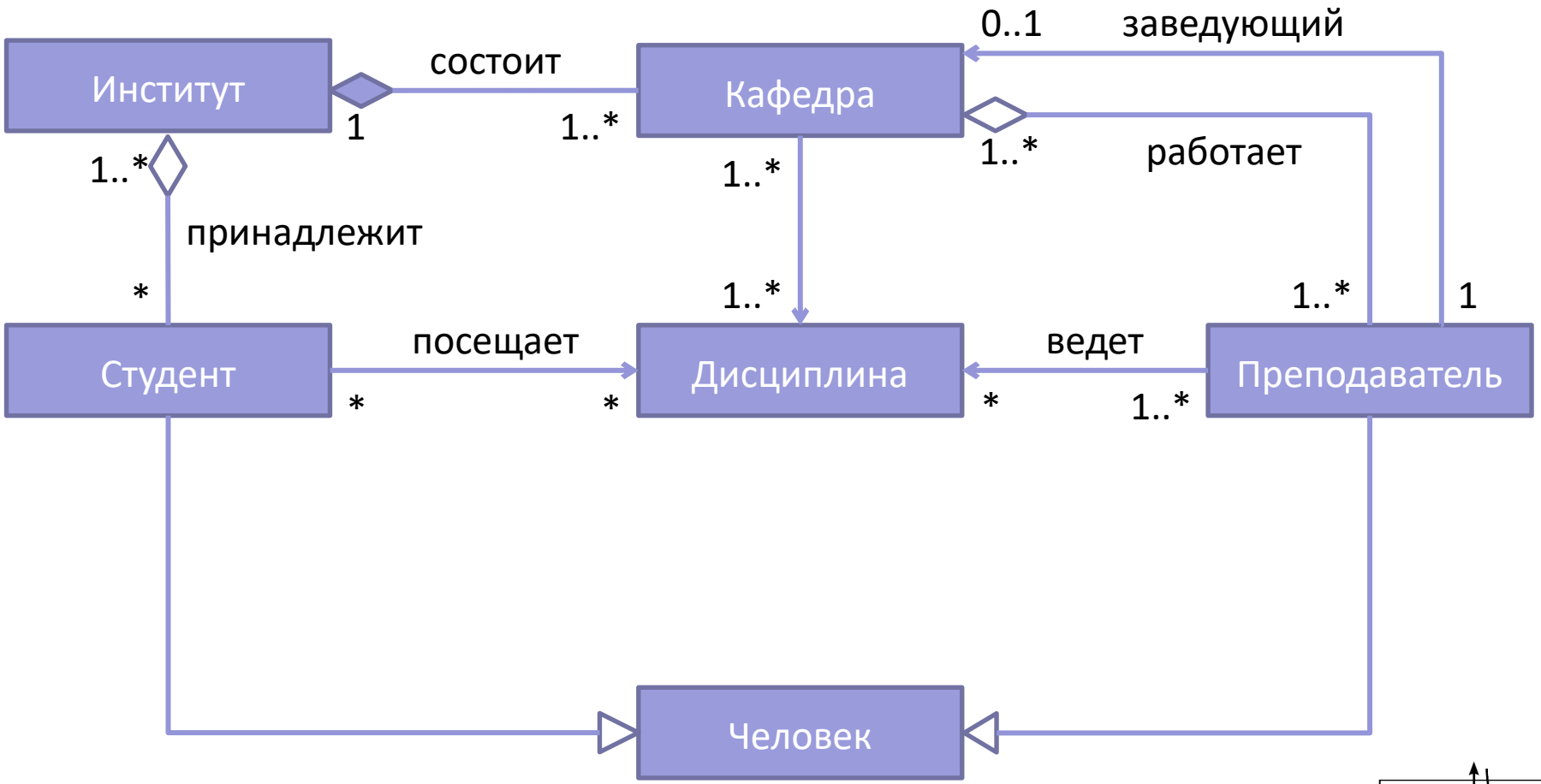


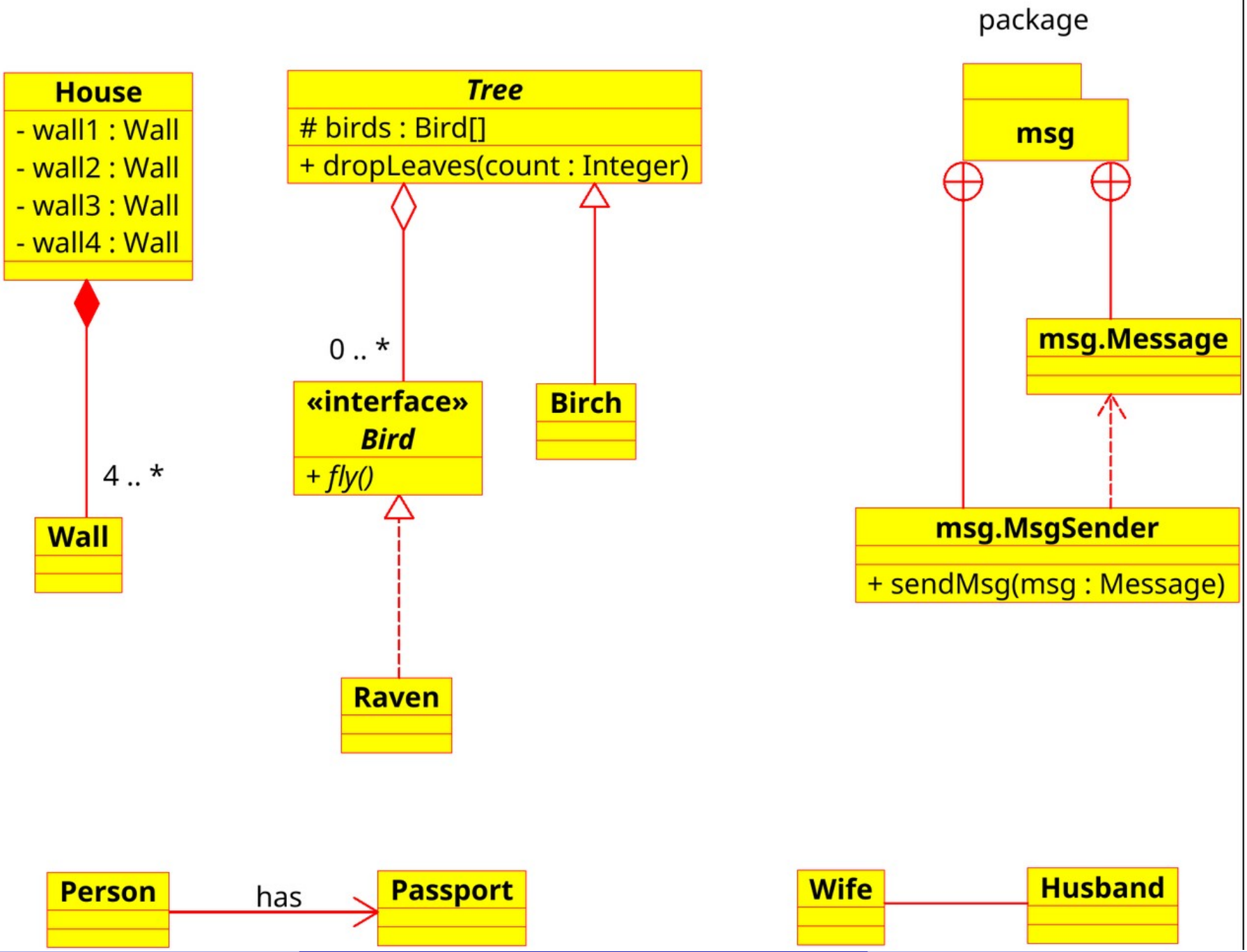
- Сущности – Классы

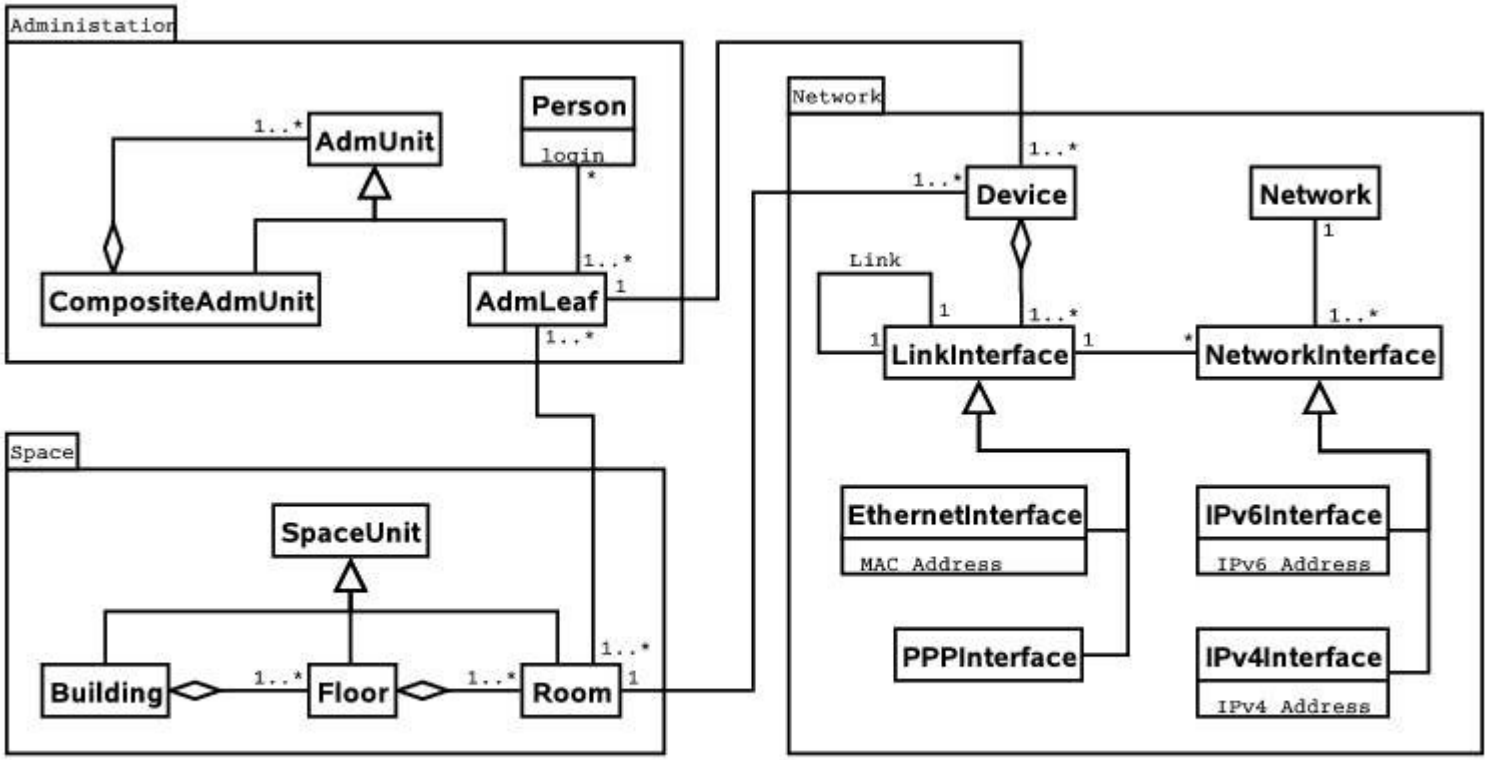
Цель - графическое представление статической структуры декларативных элементов системы (классов, типов и т. п.)

cs.petrsu.ru

- Ассоциация - показывает, что объекты одной сущности (класса) связаны с объектами другой сущности таким образом, что можно перемещаться от объектов одного класса к другому.
- Наследование(обобщение) - показывает, что один из двух связанных классов (подтип) является частной формой другого (надтипа), который называется обобщением первого.
- Реализация — отношение между двумя элементами модели, в котором один элемент (клиент) реализует поведение, заданное другим (поставщиком). Отношение целое-часть.
- Зависимость - обозначает такое отношение между классами, что изменение спецификации класса-поставщика может повлиять на работу зависимого класса, но не наоборот.
- Агрегация — это разновидность ассоциации при отношении между целым и его частями.
- Композиция — более строгий вариант агрегации (по значению).


Институт
состоит
Кафедра
0..1
заведующий
1
1..*
1..*
работает
1..*
принадлежит
1..*
*
1..*
1
Студент
посещает
Дисциплина
ведет
Преподаватель
*
*
*
1..*
Человек


House
- wall1 : Wall
- wall2 : Wall
- wall3 : Wall
- wall4 : Wall
4 .. *
Wall
Tree
birds : Bird[]
+ dropLeaves(count : Integer)
0 .. *
«interface»
Bird
+ fly()
Birch
Raven
package
msg
msg.Message
msg.MsgSender
+ sendMsg(msg : Message)
Person
has
Passport
Wife
Husband



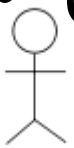
Объектная модель SAN-структуры

§7 Модели прецедентов

- Описание поведения системы с точки зрения пользователя
- Первичное моделирование системы с точки зрения выполняемых функций

=> Функциональная модель высокого уровня

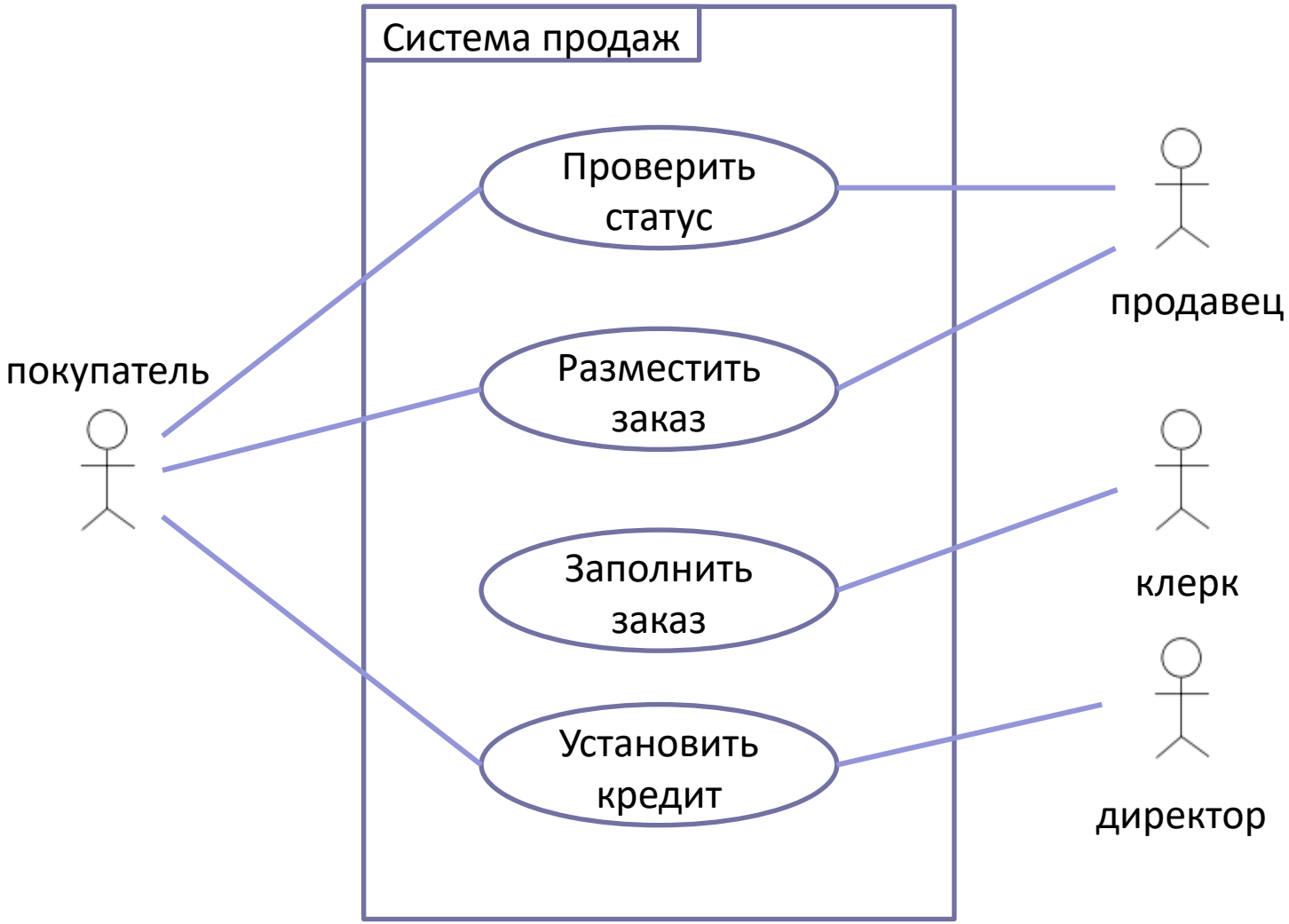
- Обозначения



Актер (источник внешних событий)

Прецедент (услуга или функция)

cs.petrsu.ru



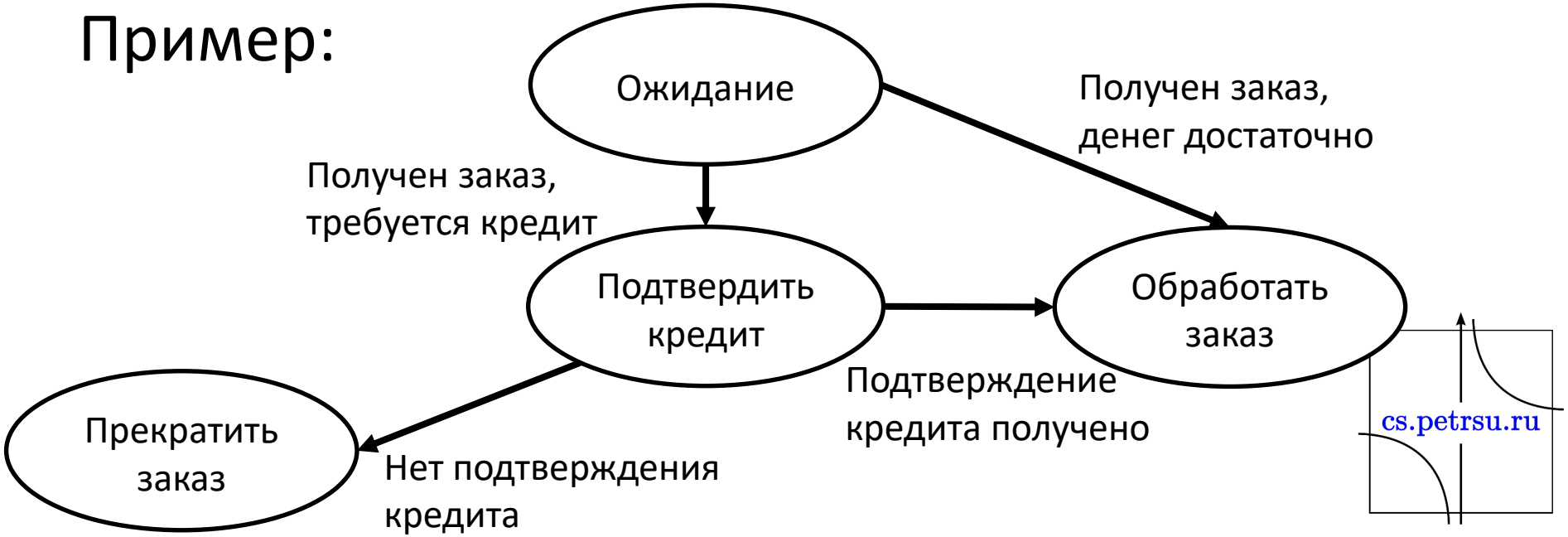
+ текстовое пояснение каждого прецедента

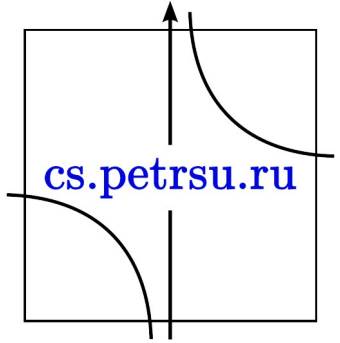
! название прецедента начинаются с глагола

§8 Модель конечных автоматов

- Система может находится в различных состояниях
- Переходы между состояниями вызываются событиями

Пример:





Проектирование

Глава №5

§1 Причины и роль проектирования

- Design ≠ project

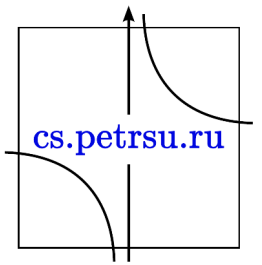
  Цикл производства ПО можно разделить на следующие 3 базовых этапа

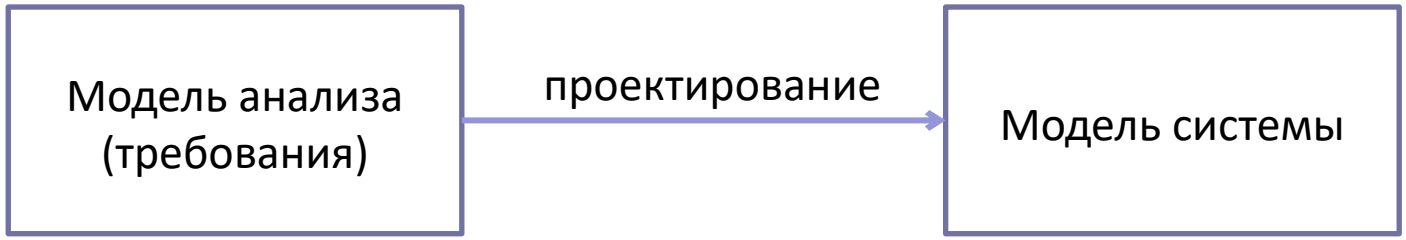
  I. Анализ (ЧТО)

  II. Синтез (КАК)

  - Проектирование (как, каким образом ПС будет реализовывать предъявленные к ней требования)
  - Кодирование
  - Тестирование

  III. Сопровождение

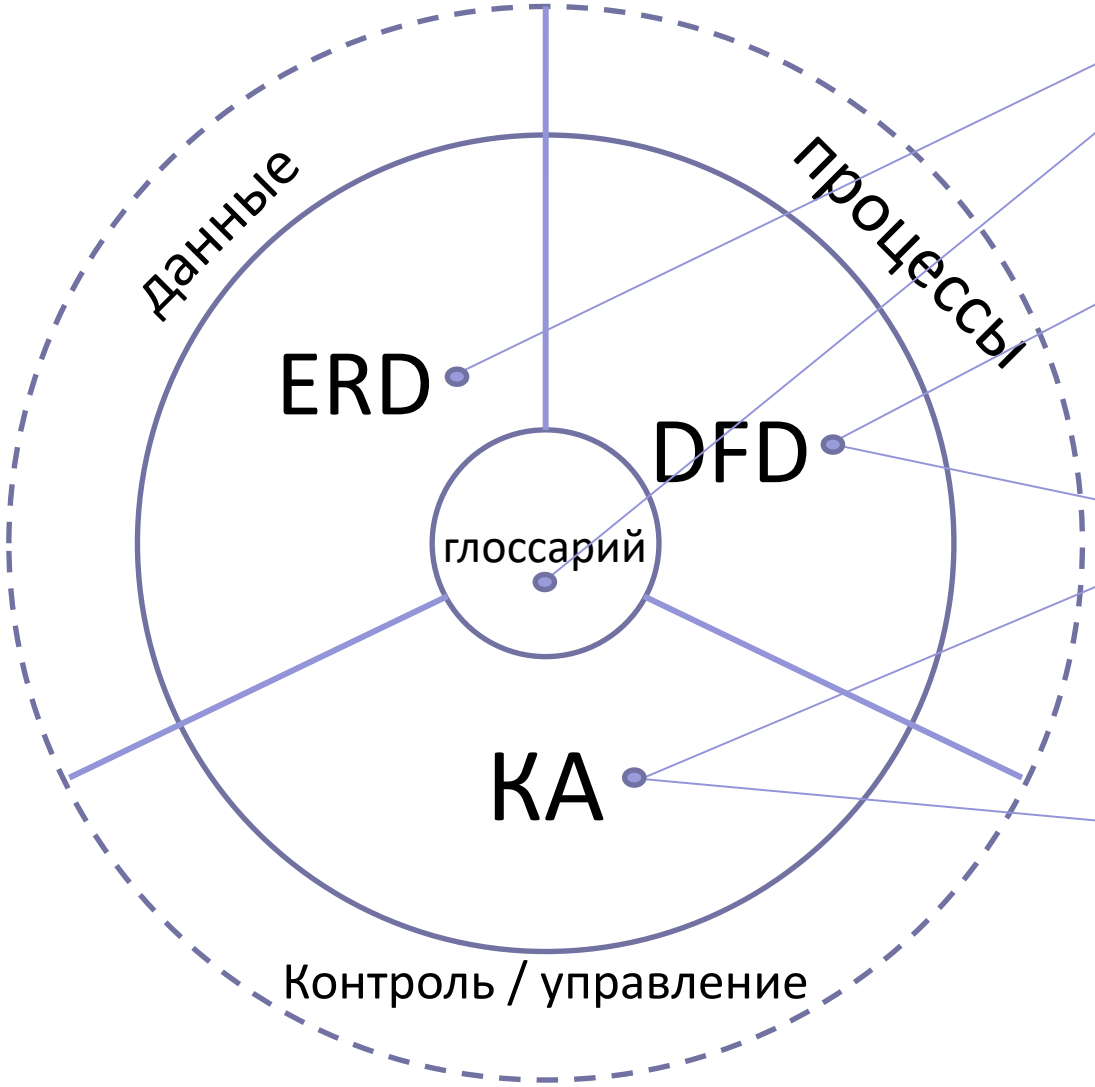




Проектирование включает

- Проектирование данных
- Проектирование архитектуры
- Проектирование интерфейса подсистем
- Проектирование интерфейса пользователя
- Проектирование подсистем
- Проектирование тестов

cs.petrsu.ru



Проектирование данных

Проектирование архитектуры

Проектирование интерфейса

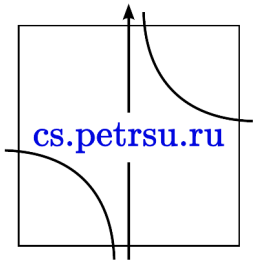
Проектирование подсистем

cs.petrsu.ru

Причина – качество!

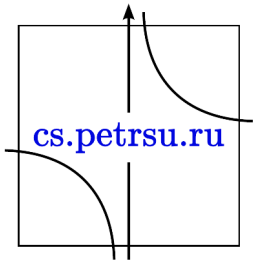
Характеристики качества проектирования:

1. Учтены все явные требования и есть учет неявных требований заказчика
2. Понятное и полное руководство для кодировщиков и тестеров
3. Всесторонняя модель ПС (данные, функциональность, поведения, эволюция)



Принципы проектирования:

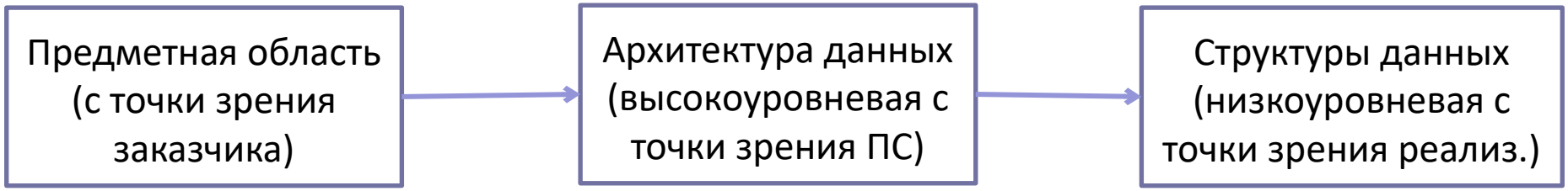
- Не использовать “взгляд через туннель” – должны быть рассмотрены альтернативы, обоснования.
- Четкая взаимосвязь с моделью требования (покрытие требования)
- Шаблоны проектирования (не изобретать велосипед)
- Структуризация (для учета возможных изменений)
- Однородность и интеграция (джентельменское поведение)
- Проектирование ≠ кодирование
- Минимизация “интеллектуальной дистанции” между ПС и поставленной задачей из практической жизни (повторение структуры предметной области)
- Оценка качества во время проектирования, а не по факту завершения



§2 Проектирование архитектуры

- Архитектура данных
- Архитектура подсистем

} архитектура ПС



Архитектура данных

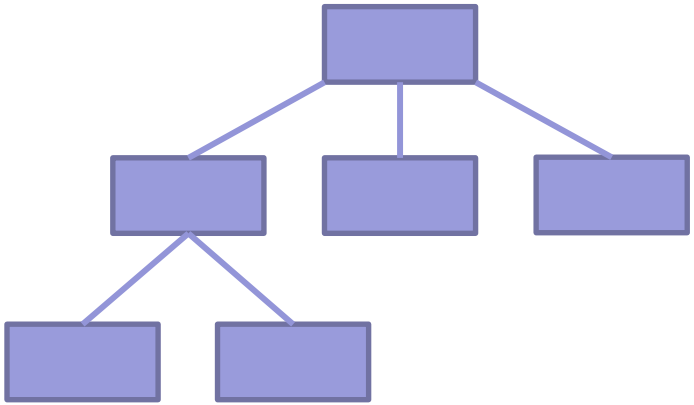
- Структуры данных
- Форматы представления данных
- Структура БД

cs.petrsu.ru

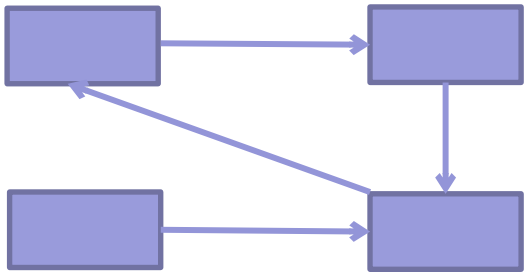
Архитектура ПС

- Подсистемы/модули/компоненты
- Взаимосвязь подсистем(ERD)
- Потоков данных между подсистемами(DFD)
- Общие структуры и форматы представления данных

- Иерархия



Сеть



Принцип модульности

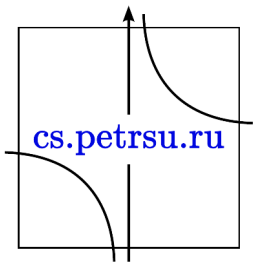
- Архитектура – это декомпозиция задачи
- Модульность – это свойство ПО, обеспечивающую интеллектуальную возможность создания сколь угодно сложного ПО

C(x) – сложность решения задач x

T(x) – затраты на реализацию x

Для 2х проблем p1 и p2:

C(p1) > C(p2) => T(p1) > T(p2) - аксиома



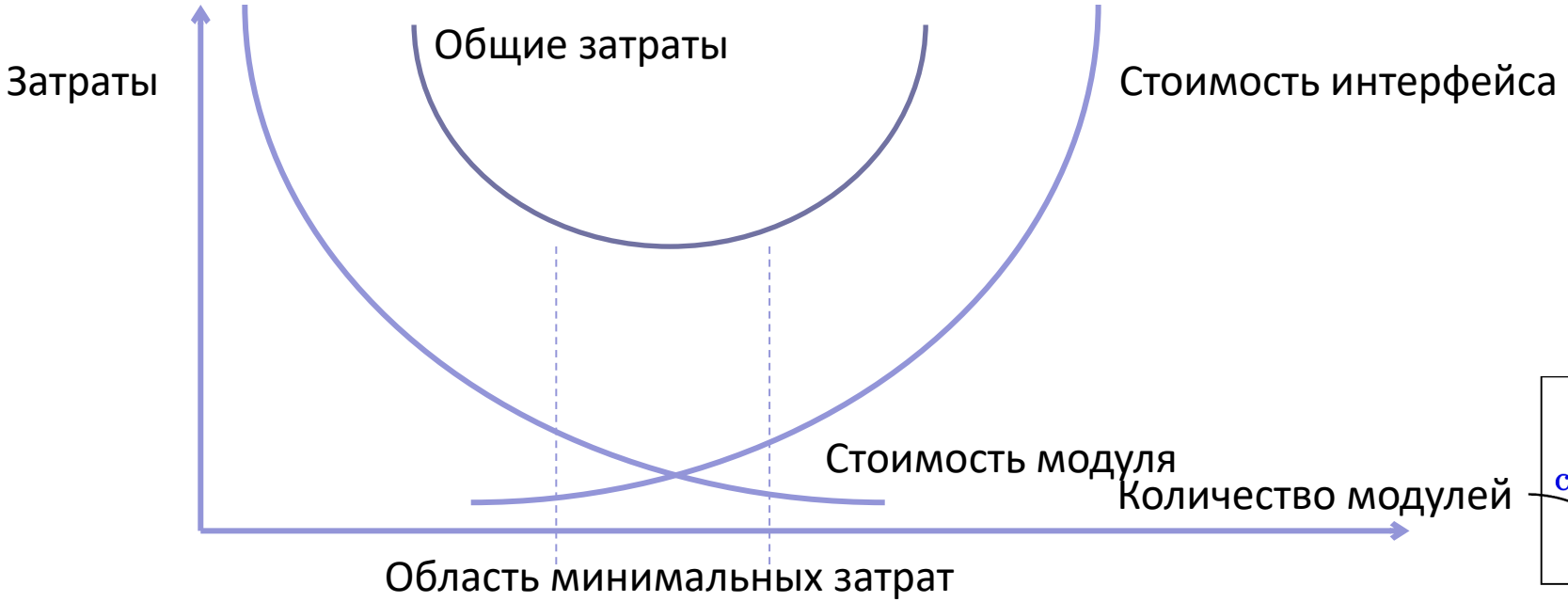
Из практики известно:

$$C(p1+p2) > C(p1)+C(p2)$$

=> Обоснование модульности:

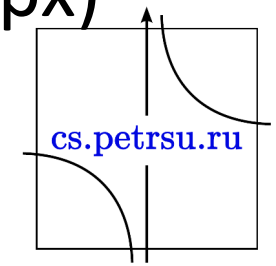
$$T(p1+p2) > T(p1)+T(p2)$$

- Однако:



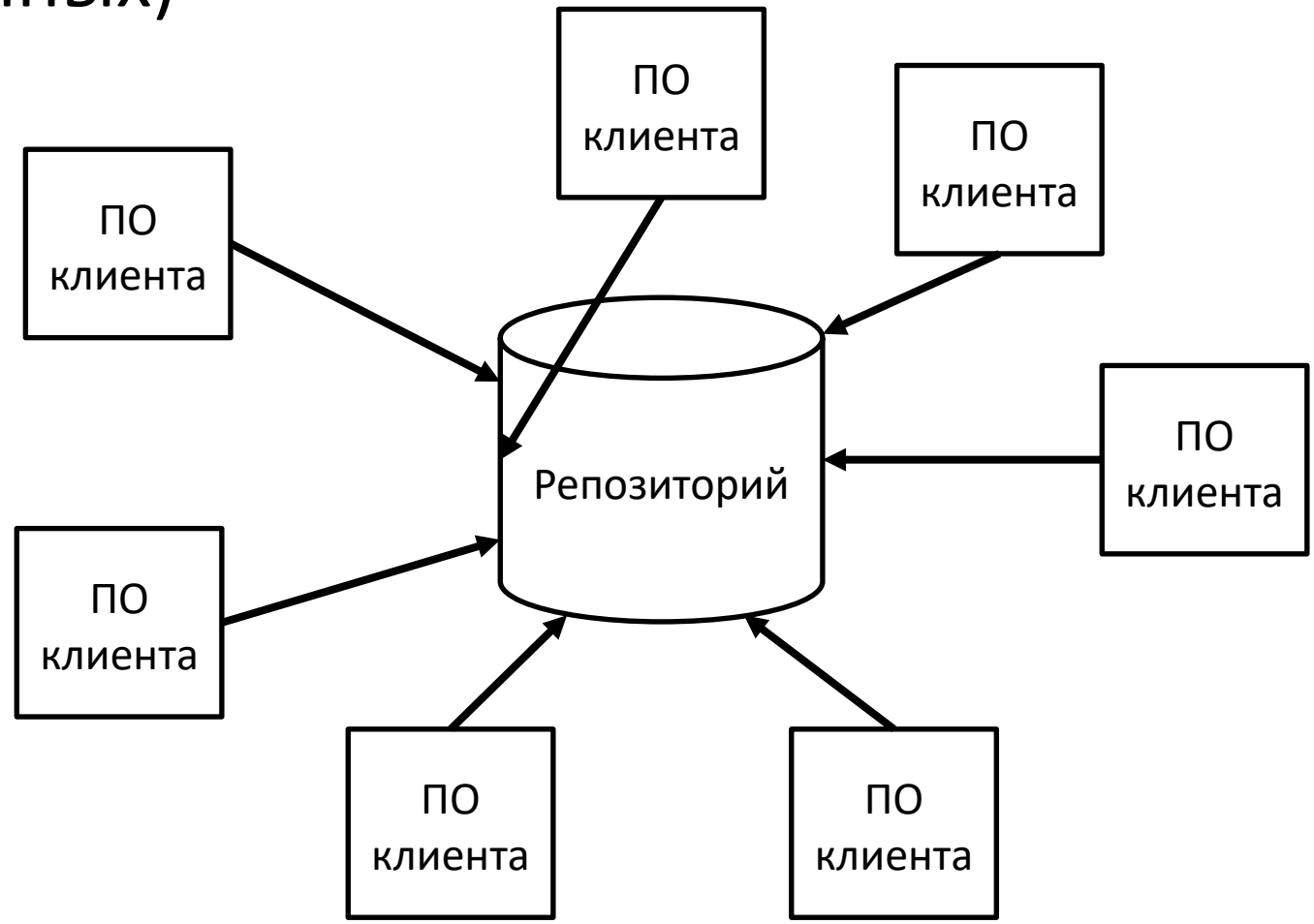
Архитектурные стили (шаблоны)

- Модель репозитория (центр.хранилице)
- Модель потоков данных
  - Конвейеры и фильтры
  - Последовательный фильтр
- Модель вызова-возврата
- Модель клиент/сервер
- Модель абстрактной машины (многоур.арх)
- Объектная модель



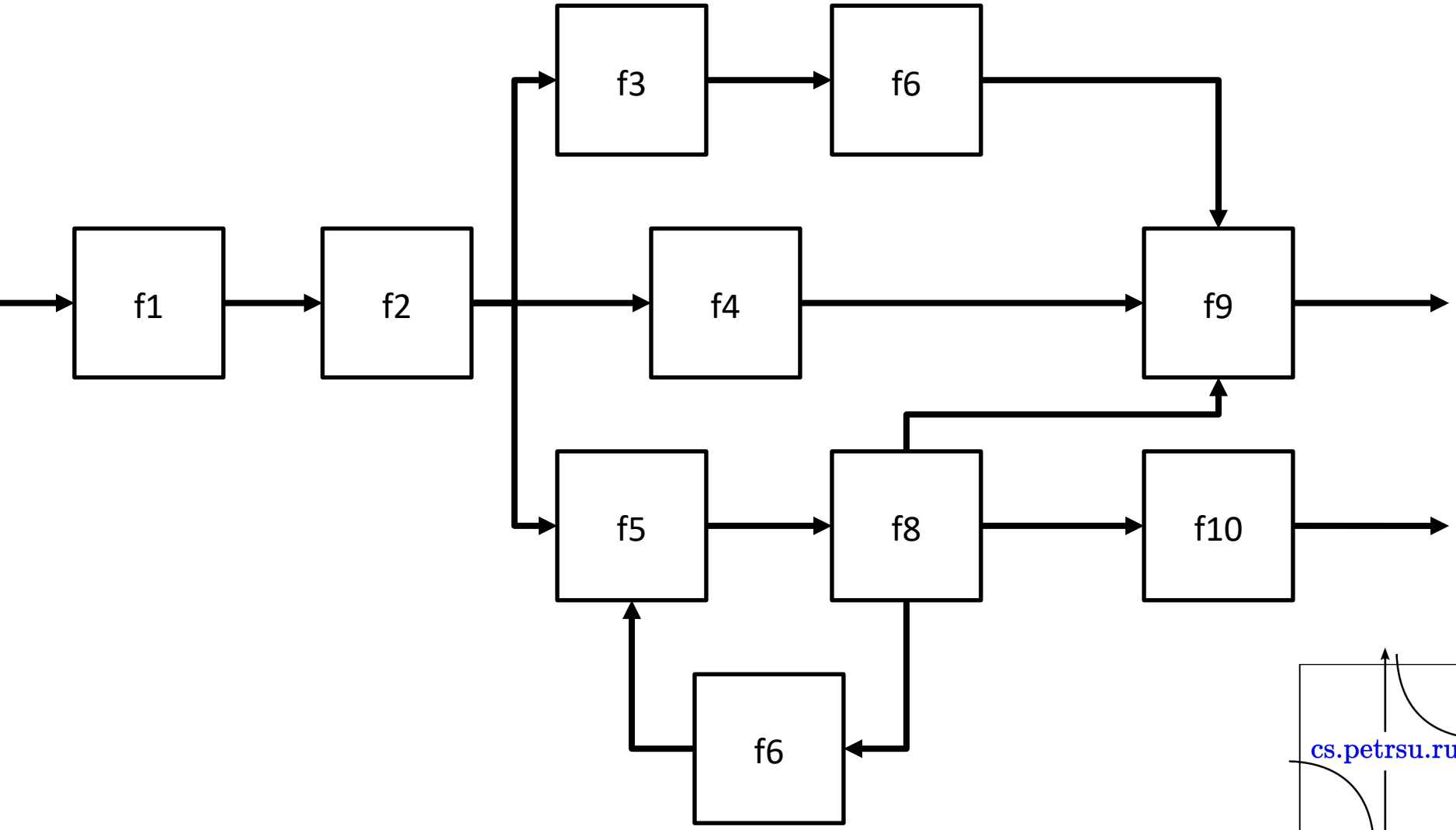
1. Модель репозитория

Основная идея - центральное хранилище данных)

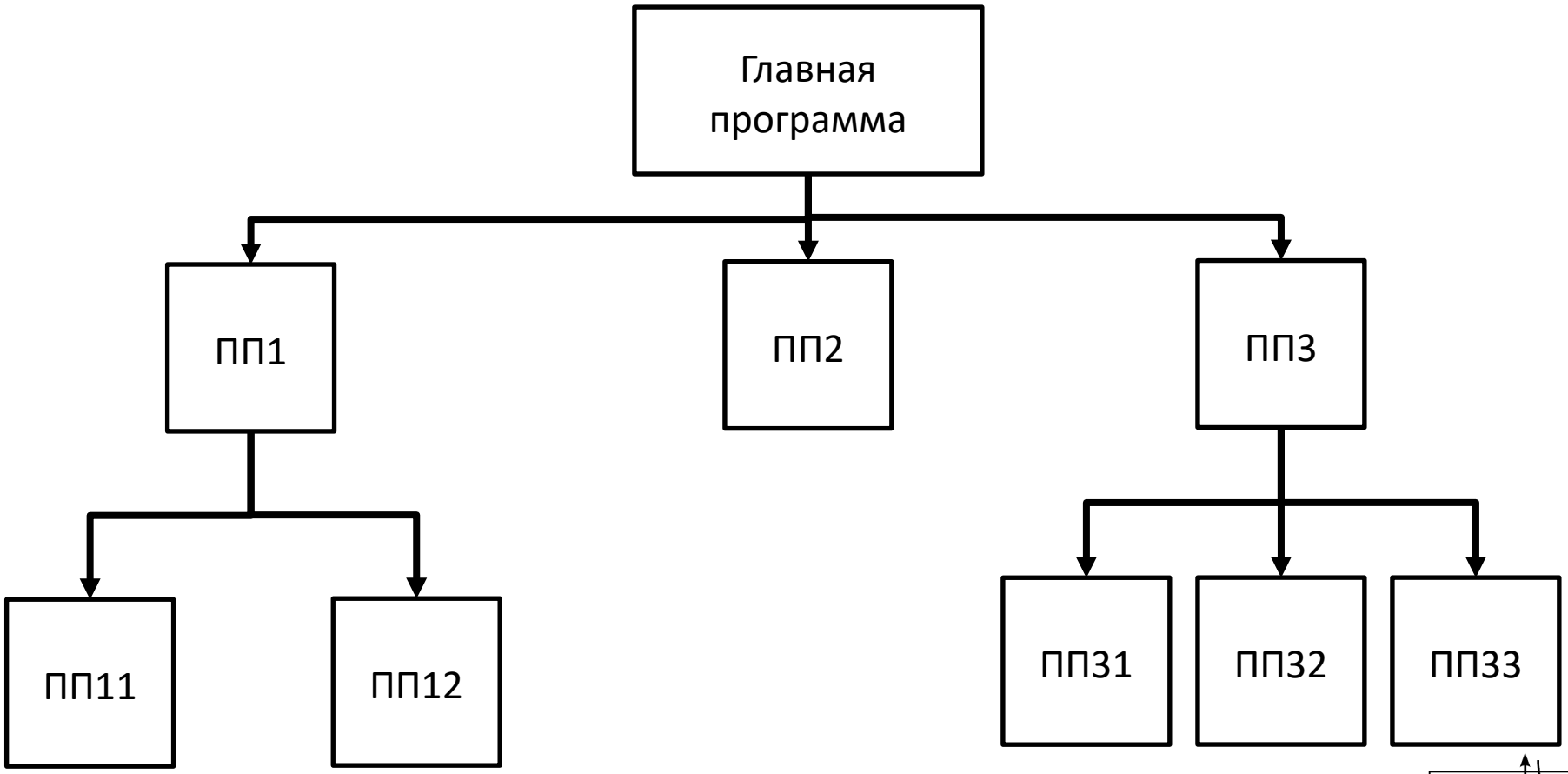


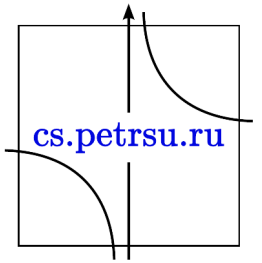
cs.petrsu.ru

2. Модель потоков данных

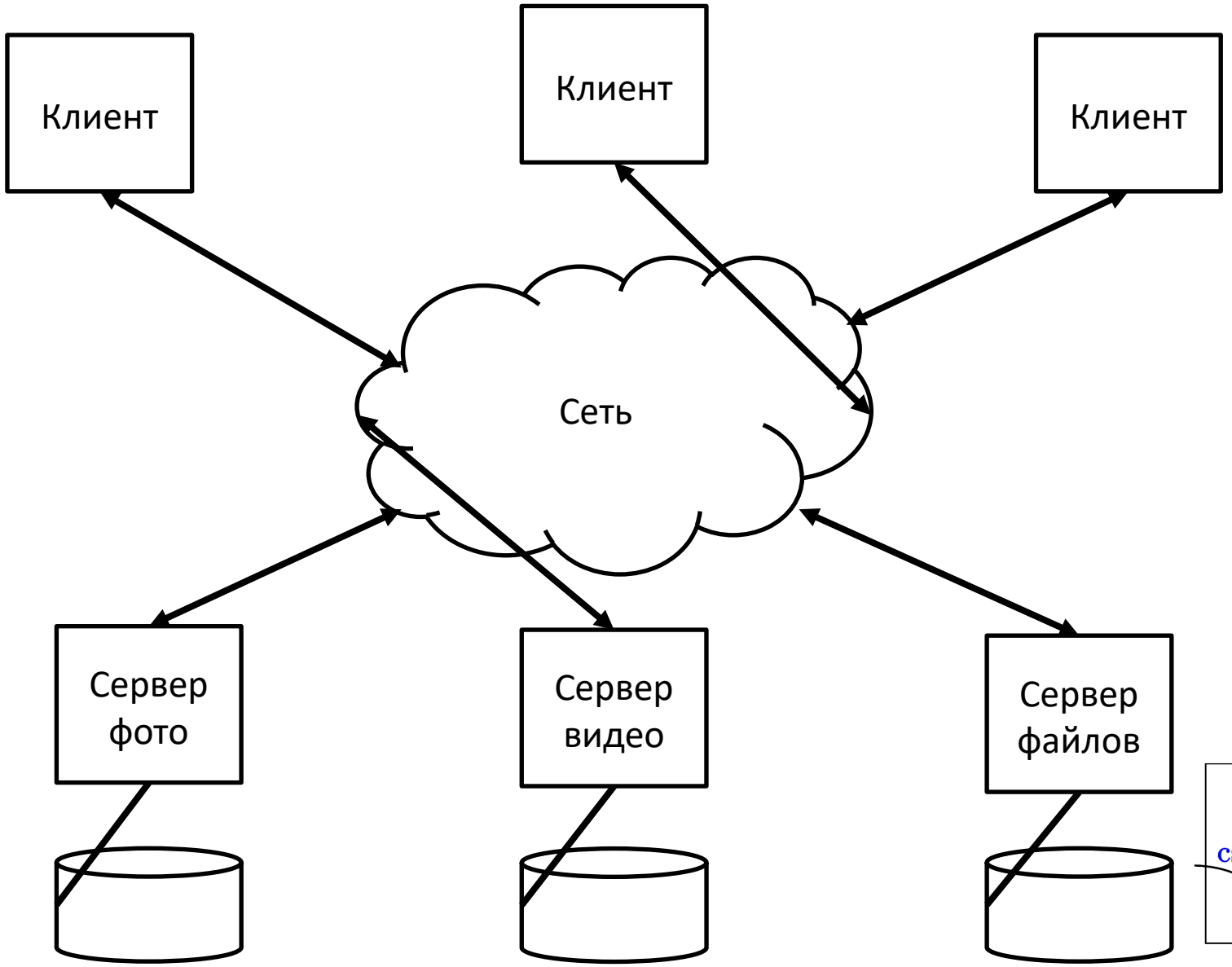


3. Модель вызова возврата





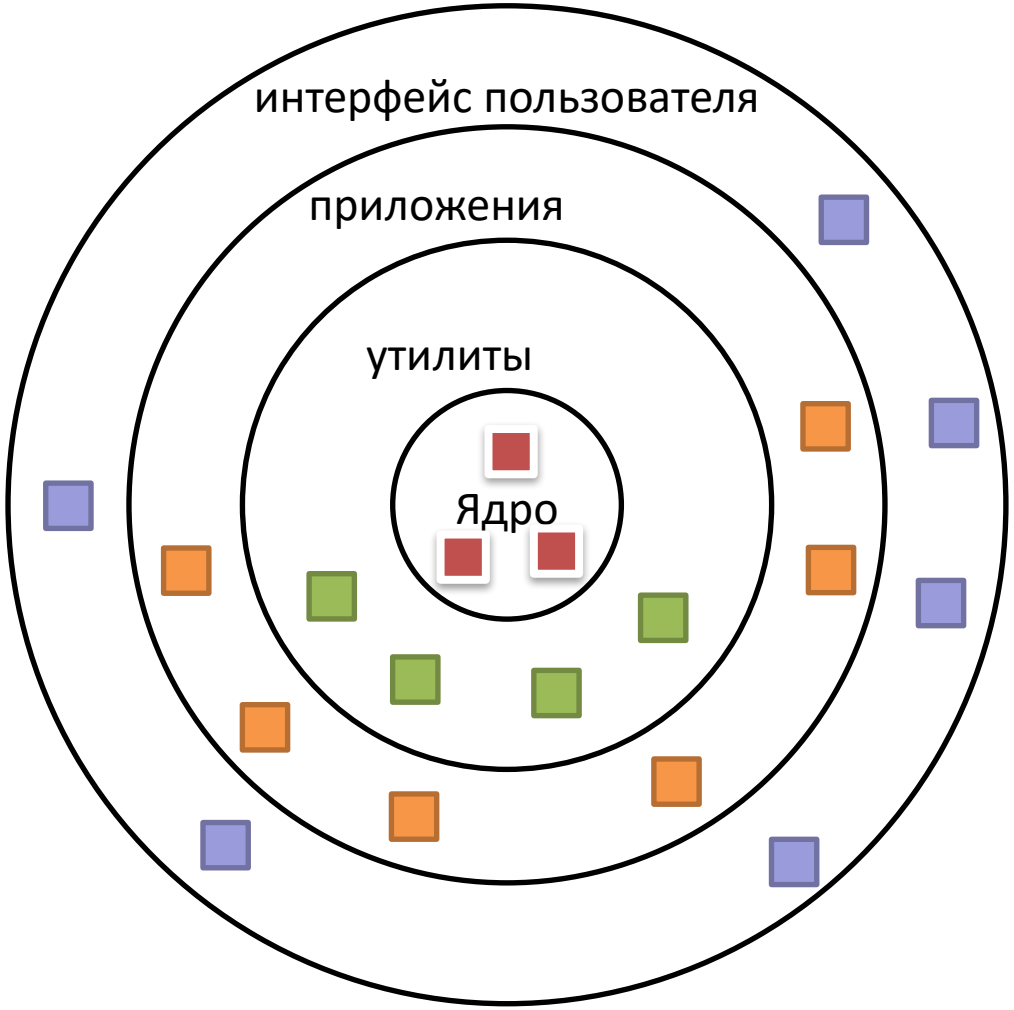
4. Модель клиент/сервер



cs.petrsu.ru

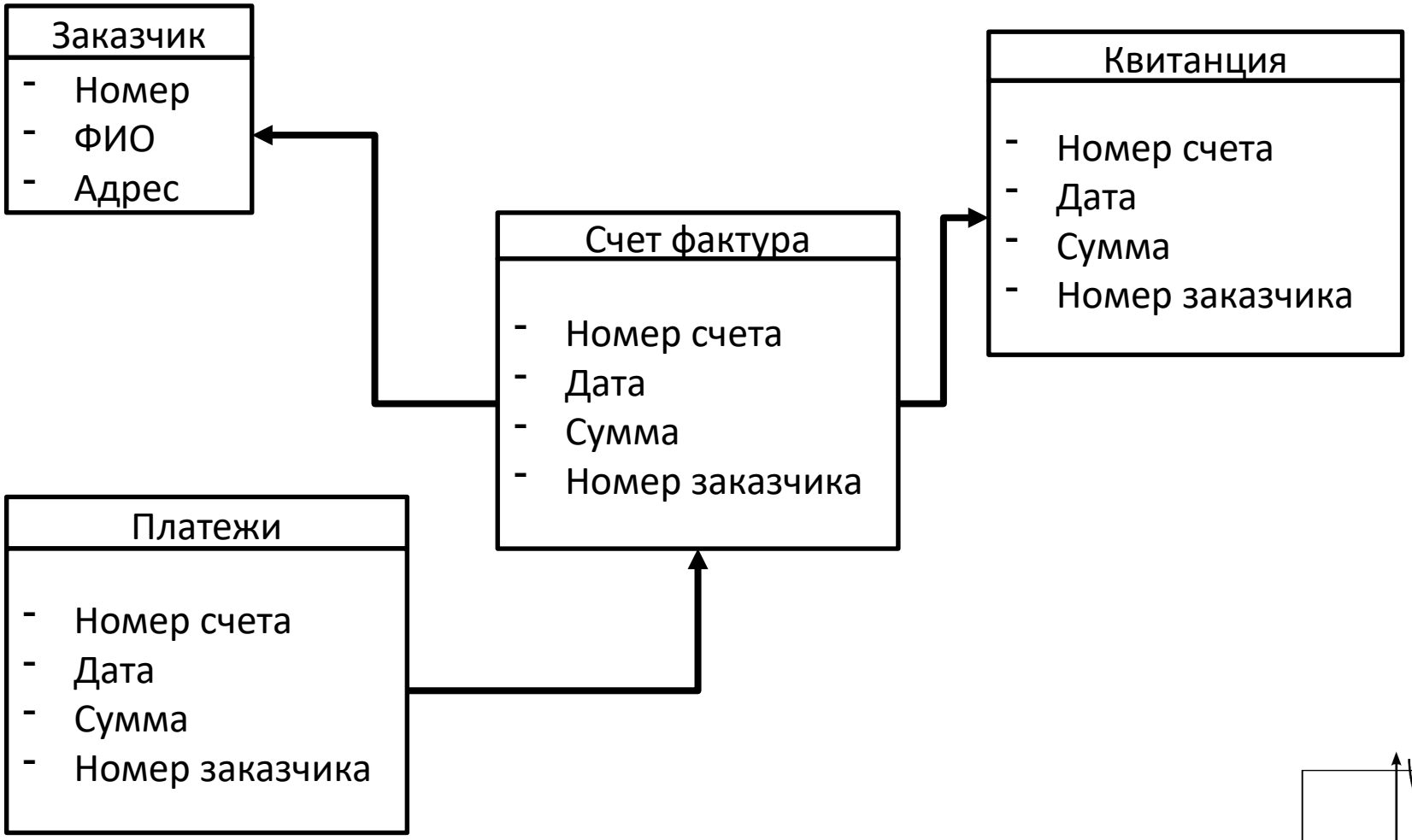
5. Модель абстрактной машины

многоуровневая архитектура



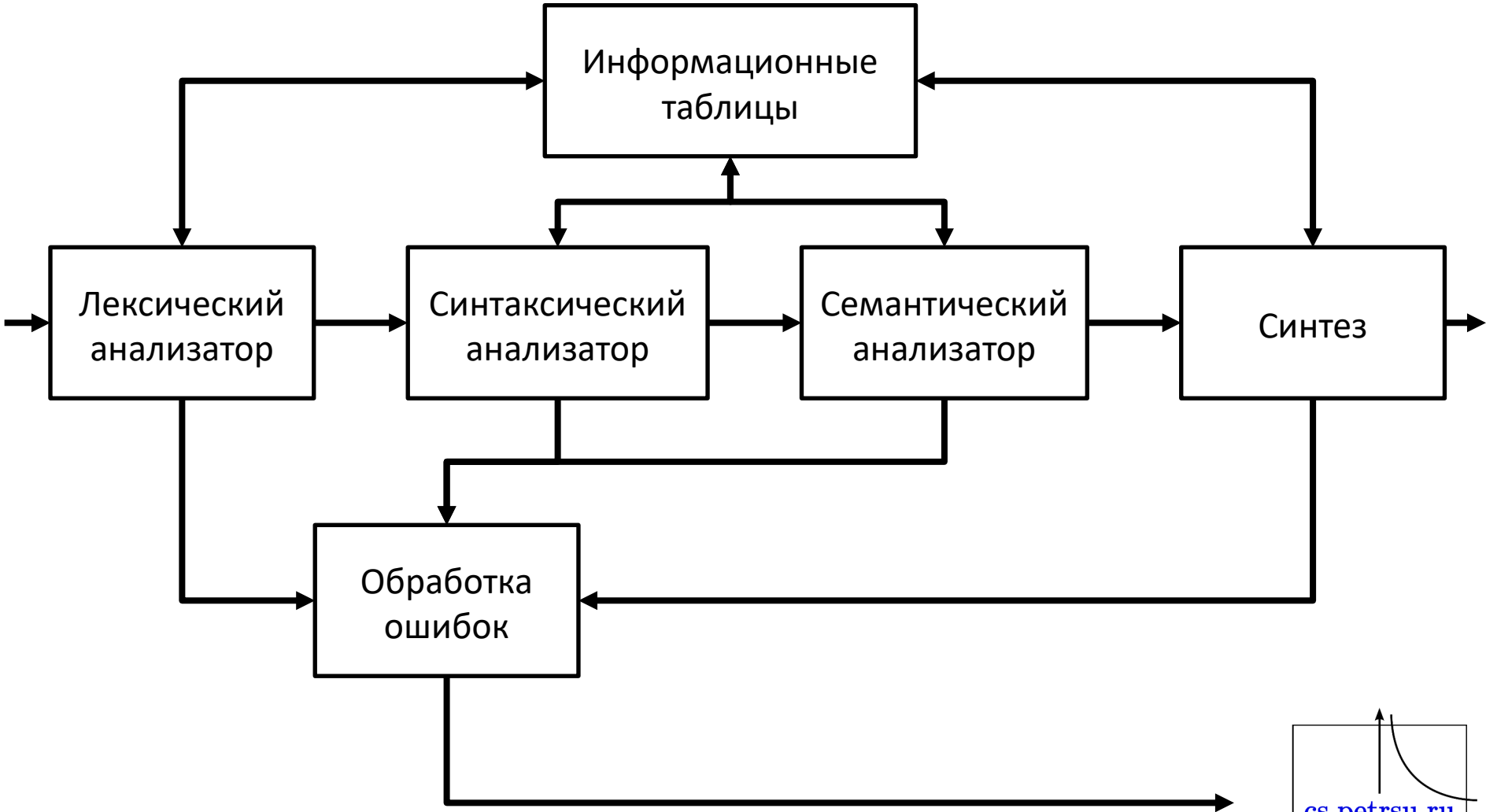
cs.petrsu.ru

6. Объектная модель



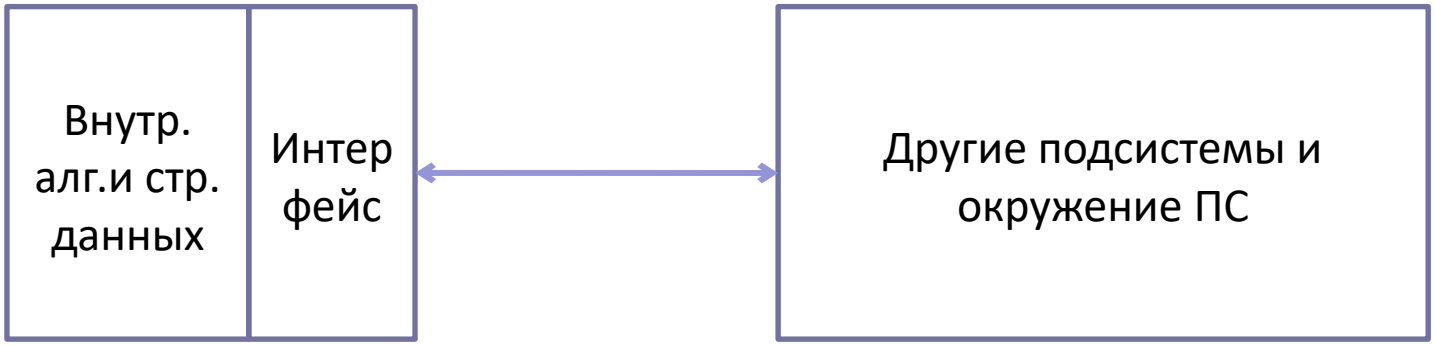
cs.petrsu.ru

7. Предметно-ориентированные модели



§3 Проектирование подсистем и интерфейсов

- Внешние и внутренне устройство подсистемы:



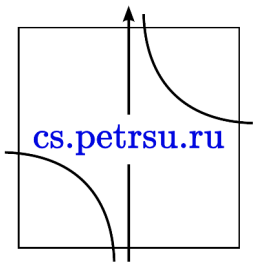
- Максимизация детализации описания
  - Очень близко к коду + логическая ясность
  - Вирт: “Алгоритм + структура данных = программа”
  - Интерфейс – четкое описание протокола взаимодействия

§4 Проектирование интерфейса пользователя

Золотые правила (Theo Mandel)

1. Пользователь центральный элемент управления
   - Не требовать от пользователя лишнего или неожиданного
   - Гибкость (выбор альтернатив на усмотрение пользователя, настройка и т.д.)
   - Undo/Cancel (возможность прервать цепочку действий в любой момент и отменить действие)
   - Скрытие технических деталей от обычного пользователя
   - Непосредственное взаимодействие с объектами на экране
   - Подтверждение деструктивных действий

2. Малая загрузка памяти пользователя
   - Чем больше помнить – тем больше ошибок
   - Уменьшение потребности в кратковременной памяти
   - Определение осмысленных установок по умолчанию
   - Интуитивные сокращения и обозначения
   - Структуризация (четкая иерархия)



3. Согласованность
   - Унификация (однородность, однотипность) схожесть команд, меню и т.д.
   - Проверка граничных значений и др. виды корректности
   - Согласованность навигации по задачам

Инженерная психология, 3 модели пользователей:

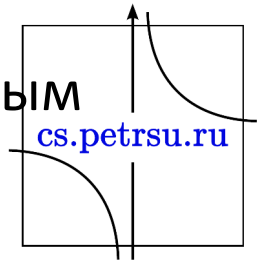
- Новички
- Стандартный
- Продвинутый

Оценка интерфейса

- Виды представления информации (формы) – визуальный
- Сообщения в ответ на действия пользователя
- Диалоговая справочная система
- Руководство пользователя

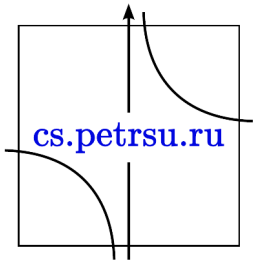
Количественная оценка

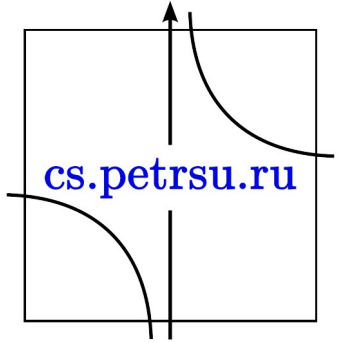
- Изучаемость (количество времени обучения для начала продуктивной работы)
- Скорость работы (скорость реакции системы)
- Устойчивость (к ошибкам пользователя)
- Восстанавливаемость (после ошибки пользователя)
- Адаптируемость (способность “подстроиться” к разным стилям работы пользователя)



Методики оценивания

- Анкеты пользователей
- Наблюдение за пользователем, с обсуждением его способов использования системы
- Видеонаблюдение типичного использования системы
- Добавление кода





Верификация и аттестация

Глава №6

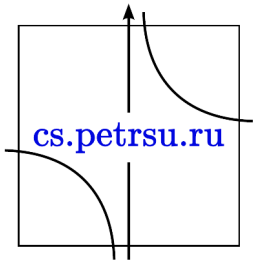
Верификация и аттестация – это процесс проверки и анализа в ходе которых проверяется соответствие результатов и требований

- Верификация – правильно ли создана система (соответствие спецификации)
- Аттестация – правильно ли работает система (соответствие ожиданиям заказчика) – более общий процесс

Основные методики:

- Инспектирование
- Тестирование

} => Отладка



§1 Методики

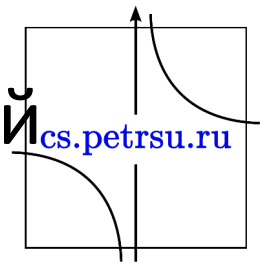
1. Инспектирование

Анализ и проверка различных представлений ПС (например, спецификация требований, архитектура, исходный код)

- ! Не требует исполнения ПС

  $\Rightarrow$ статический метод

- Исполнение ПС – зачастую дорогостоящий процесс



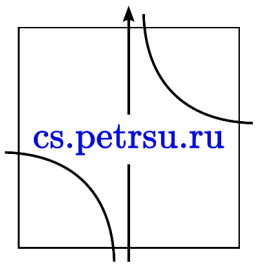
a) За один сеанс инспектирования можно обнаружить несколько ошибок, а при тестировании как правило одну.
b) Сосредоточено на конкретном типе ошибок

Объекты инспектирования:

- Документу и модели
- Код программы

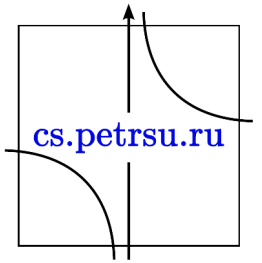
Дополняющие методы

- Автоматический статический анализ кода
- Формальные методы



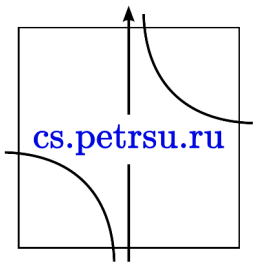
Инспектирование – формализованный процесс

- Роли:
  - Автор – ответственное за объект лицо,
  - Инспектор – осуществляет проверку,
  - Рецензент – “озвучивает” объект проверки на собрании,
  - Секретарь – протоколирование собраний инспектирования,
  - Председатель – управление и организация инспектирований



- Узкая тематика: отдельный объект, тип ошибок
- Длительность процесса:
  - 1-2 часа на подготовку к инспектированию
  - 1 час на собрание инспекционной группы

- ! Более 60% ошибок в ПО можно обнаружить в процессе инспектирования кода
- ! Более 90% ошибок в ПО можно обнаружить если использовать инструменты автоматизации и математические методы

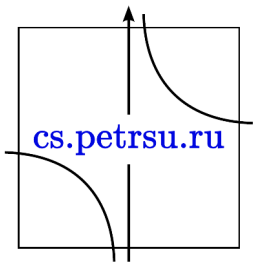


2. Тестирование

Процесс инспектирования (статический метод) не заменяет тестирования (динамический метод)

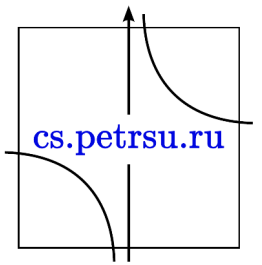
- Проверка:
  1. Правильность внутренней логики компонент ПО
  2. Правильность работы на различных входных данных
- Результат:
  – ошибки в функциональности, поведении и т.д.

Тестирование не деструктивный процесс!



- Свойства:
  - Это процесс выполнения ПО для нахождения ошибки
  - Успешный тест, если найдена ошибка
  - Хороший тест – с большой вероятностью находит не обнаруженную ранее ошибку

Тестирование не может показать отсутствие ошибок и дефектов, оно лишь показывает наличие ошибок и дефектов.



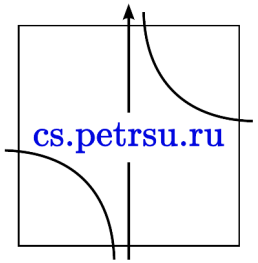
3. Отладка

- Локализация и устранение ошибок
- Следствие тестирования – отладка устраняет найденные ошибки
- Журнал выполнения тестирования

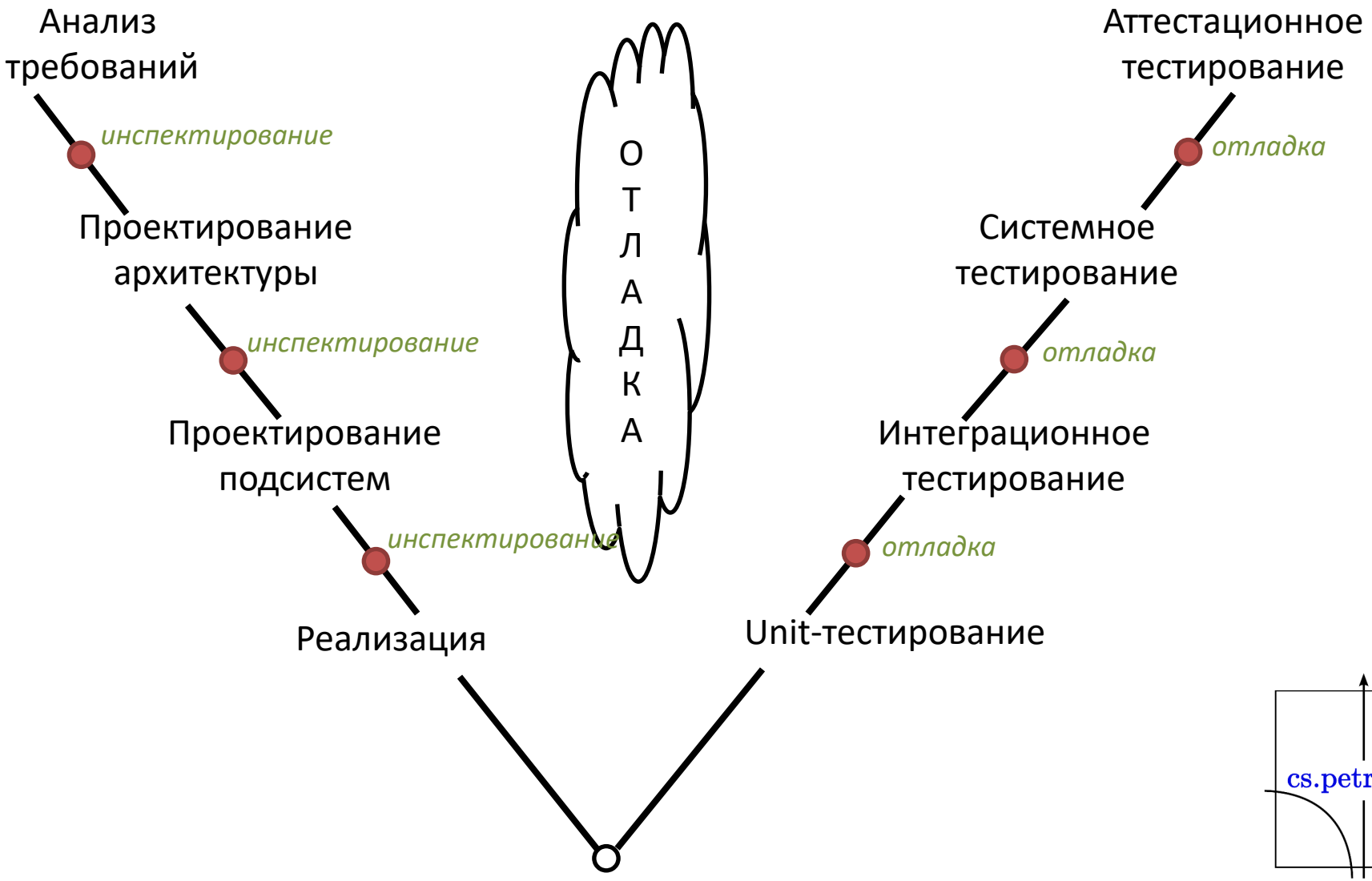
Дата проведения теста	Тестировщик	Краткое описание выполненного теста	Результат
17.01.2020	Иванов И. И.	Например: № теста(ссылка на план тестирования) и входные данные	- Ошибок не обнаружено - Ошибка обнаружена, дается описание и требуемые действия

! Управление тестированием

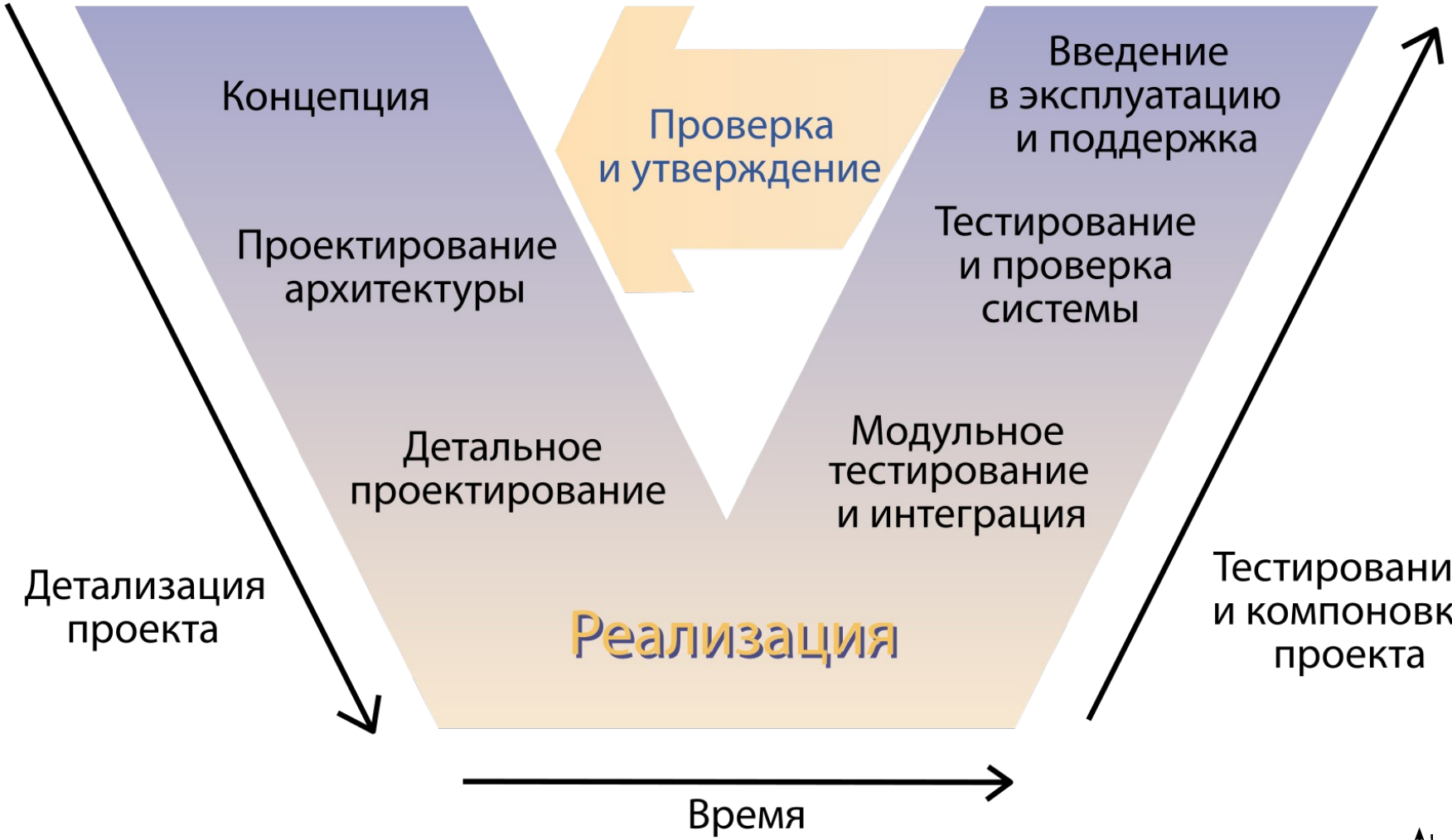
- Все тесты должны быть успешно пройдены
- Отслеживание отладки для не пройдённых тестов
- Регрессионное тестирование



4. V-модель



cs.petrsu.ru

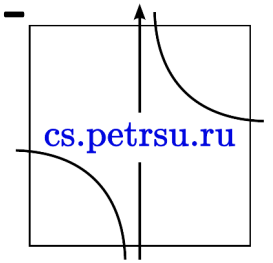

Концепция
Проектирование архитектуры
Детальное проектирование
Проверка и утверждение
Введение в эксплуатацию и поддержка
Тестирование и проверка системы
Модульное тестирование и интеграция
Детализация проекта
Реализация
Тестирование и компоновка проекта
Время

cs.petrsu.ru

Концепция V-образной модели была разработана Германией и США в конце 1980-х годов независимо друг от друга:

- Немецкая - аэрокосмическая компания IABG для Министерства обороны
- Американская (VEE) - национальный совет по системной инженерии для спутниковых систем

Современной версией V-Model является V-Model XT - утвержденная в 2005 году



Цели V-Model:

- Минимизация рисков:

  Проект прозрачный и повышается качество контроля проекта путём стандартизации промежуточных целей, результатов и ответственных лиц. Позволяет выявлять отклонения и риски на ранних стадиях и улучшает качество управления проектов.

- Повышение и гарантии качества:

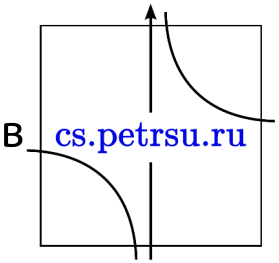
  Стандартизованная модель разработки позволяет добиться результатов желаемого качества. Промежуточные результаты проверяются на ранних стадиях. Универсальное документирование облегчает читаемость, понятность и проверяемость.

- Уменьшение общей стоимости проекта:

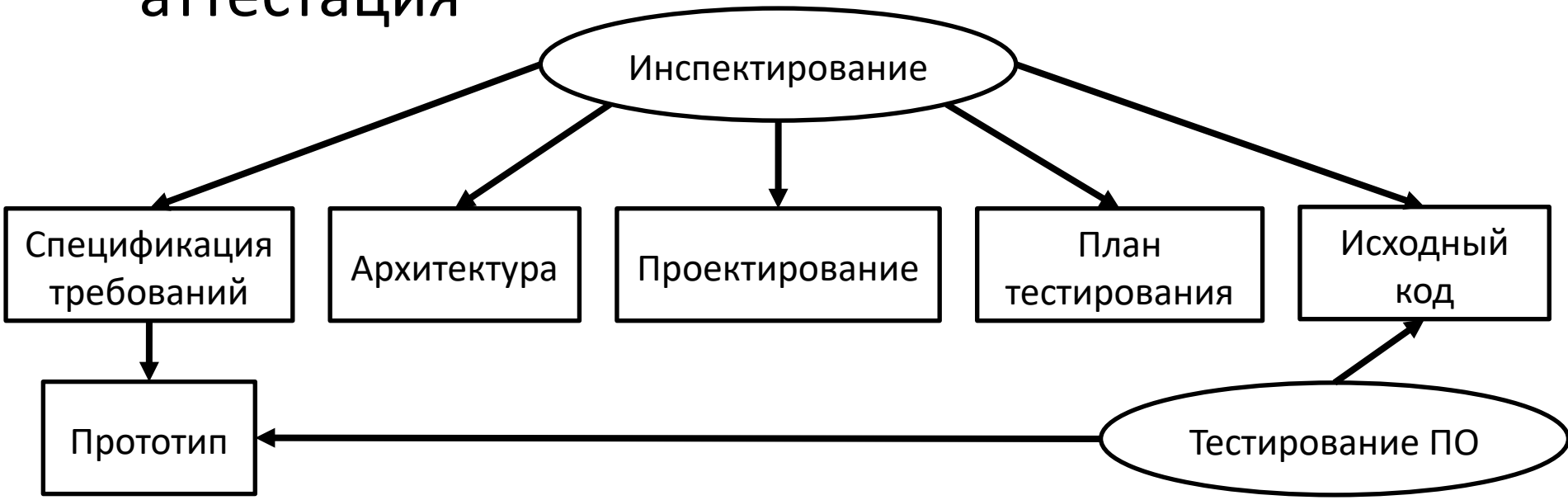
  Ресурсы на разработку, производство, управление и поддержку - заранее просчитаны и проконтролированы. Получаемые результаты универсальны и легко прогнозируются.

- Повышение качества коммуникации между участниками проекта:

  Универсальное описание элементов и условий облегчает взаимопонимание участников проекта. Уменьшаются неточности в понимании между пользователем, покупателем, поставщиком и разработчиком.



- Статическая и динамическая верификация и аттестация



- Инспектирование применимо всегда
- Тестирование – только исходный код и прототип

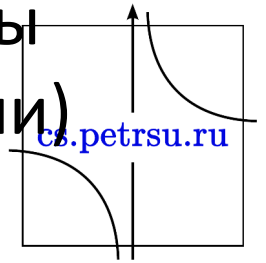
! Статическими методами проверяют только соответствие спецификации, а не правильность функционирование ПС

! Нельзя проверить многие нефункциональные требования

§2 Разработка вариантов тестов

1. Метод черного ящика
   – Зная что ПО должно делать, демонстрируем это на соответствующих примерах (близко к требованиям – проверка требований)

2. Метод белого ящика (прозрачного)
   – Зная внутреннюю структуру ПО (алгоритмы), проверяем правильность логики программы (проверка проектных решений и реализации)



Принципы тестирования:

- Тестирование – организационный процесс
  - План тестирования составляется задолго до выполнения тестирования
  - Критерии аттестации – на этапе анализа требований
  - Детальное описание тестов – на этапе планирования
- От малого к большому
  - Критерии аттестации -> детальные тесты (черный ящик)
  - Блочное тестирование (отдельные элементы) -> интеграционное тестирование -> системное тестирование
- Принцип “Парето”: 80% ошибок в 20% компонент
- Покрывающее тестирование
  - Всеобъемлющее тестирование не возможно
  - Разбиение тестов на классы
- Независимые тестировщики
  - Группа разработчиков отделена от группы тестировщиков

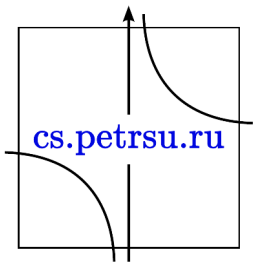
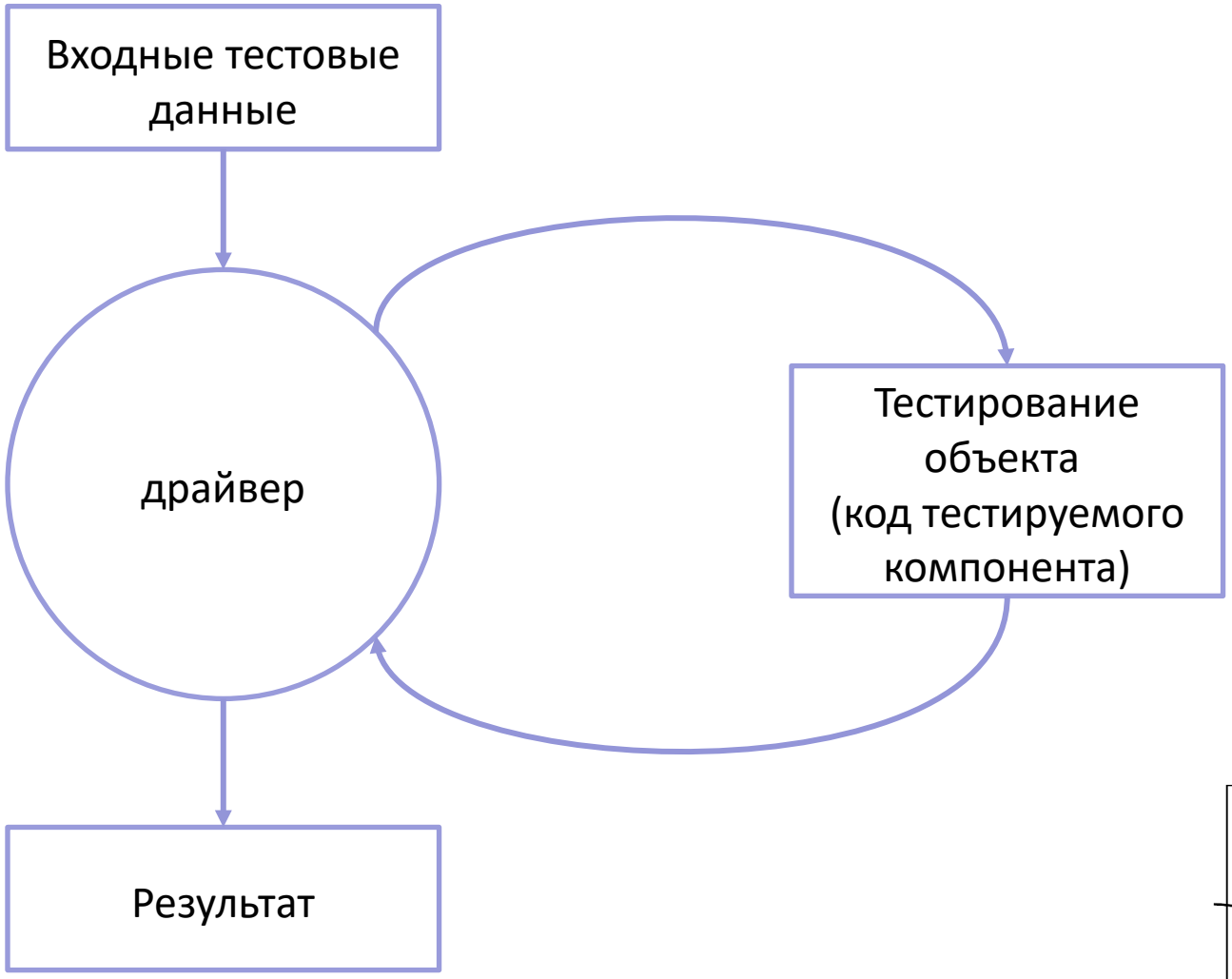
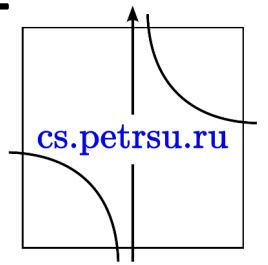


Схема тестирования



§3 Тестирование методом белого ящика

- Тесты создаются на основе структуры ПС
- Все независимые пути выполнения программы
- Все ветви if-then-else
- Все циклы
- Внутренние структуры данных
- Применяется для небольших компонент

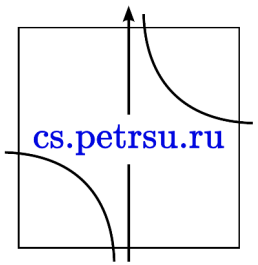


1. Тестирование ветвей (путей выполнения программы)

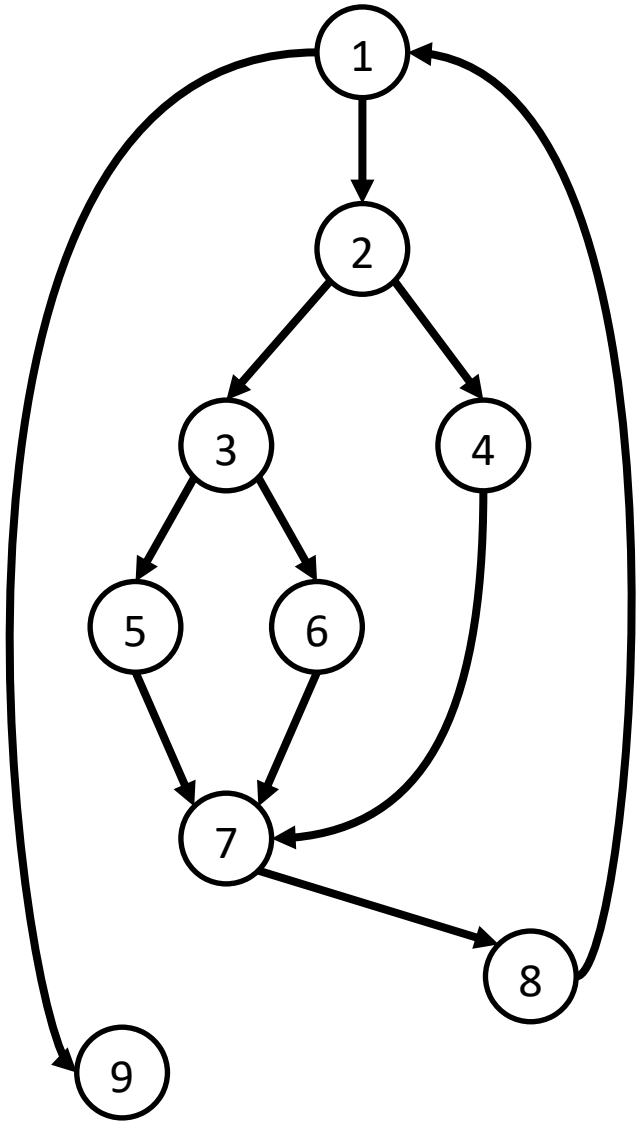
Цель: разработка такого набора тестов, чтобы программа прошла через каждый путь хотя бы один раз хотя бы в одном из тестов.

Т.е. хотя бы однократное выполнение каждого оператора программы

Обнаружение и тестирование базовых путей



Построение потокового графа:



2. Тестирование циклов

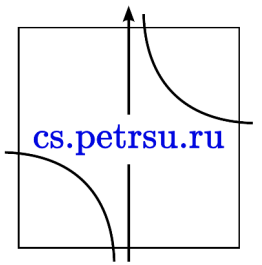
Цель: разработка такого набора тестов, чтобы покрыть все возможные варианты на повторение цикла

N – максимальное число выполнений

M – реальное число выполнений

- Пропуск цикла (M=0)
- M=1 (одноразовое выполнение)
- M=2
- M – типичное число выполнений (2<M<N)
- M=N-1
- M=N
- M=N+1

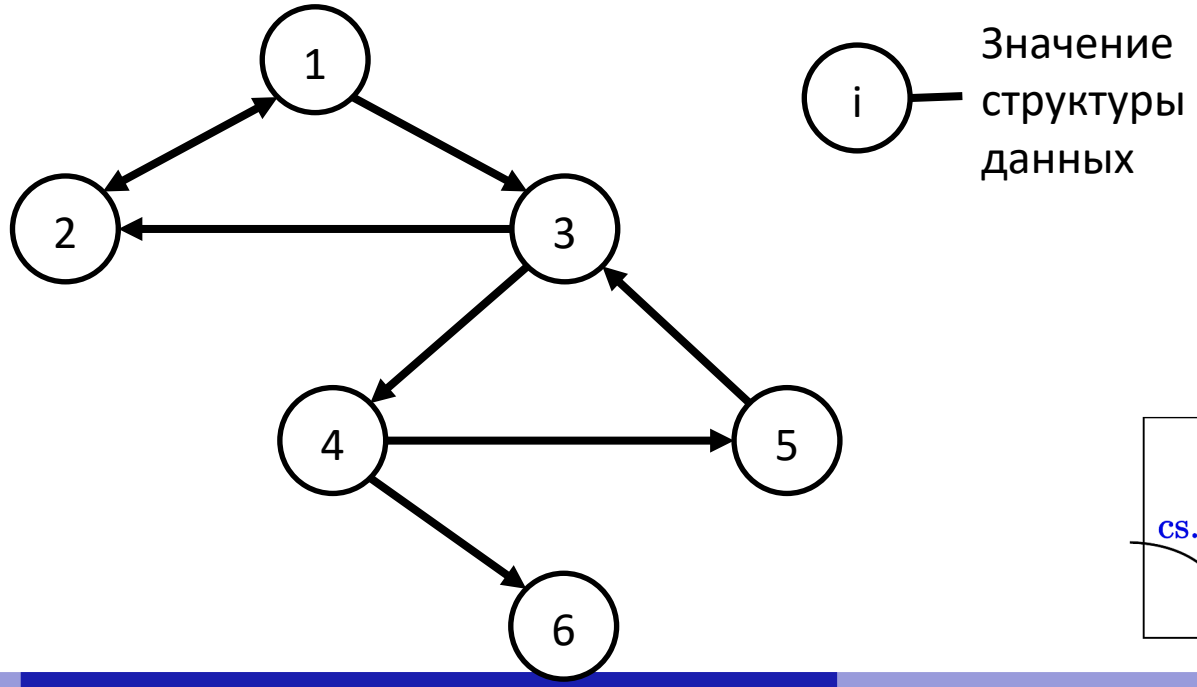
Граничные значения



3. Тестирование состояний

- Состояние программы – значение всех (или некоторых) переменных программы

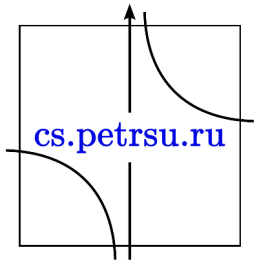
Цель: разработка такого набора тестов, чтобы программа прошла через каждое состояние хотя бы один раз в одном из тестов



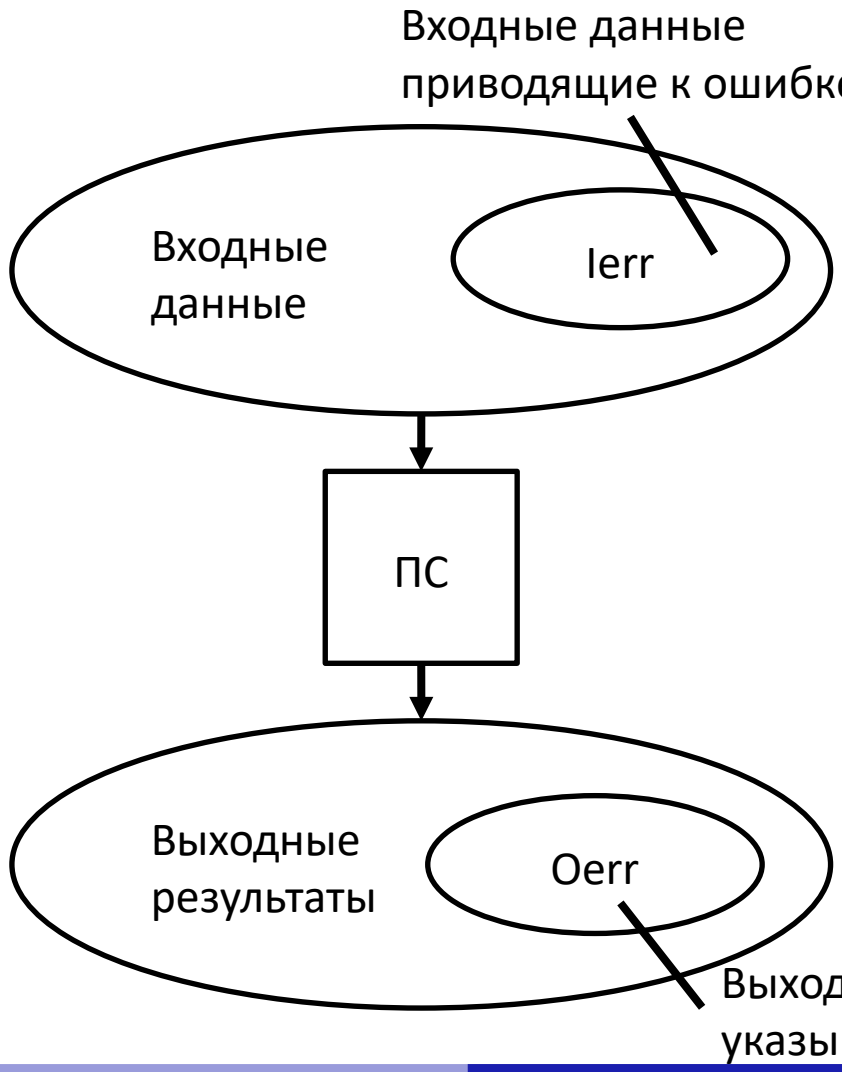
cs.petrsu.ru

4. Другие способы тестирования

- Тестирование базовых путей
- Тестирование условий
- Тестирование потоков данных
- …

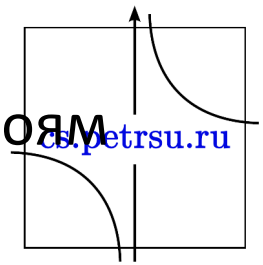


§4 Метод черного ящика
(функциональное тестирование)



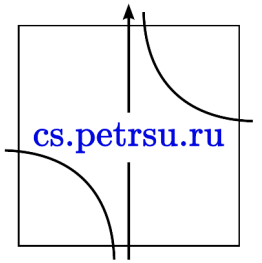
Не важно как устроена система, важно чтобы правильно выполнялась функциональность

- Функциональность
- Поведение и производительность
- Специфика работы для определенных классов входных данных
- Секретность
- Устойчивость к сбоям



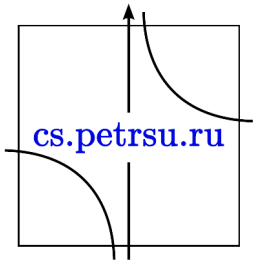
1. Разбиение входных данных по эквивалентности

- .



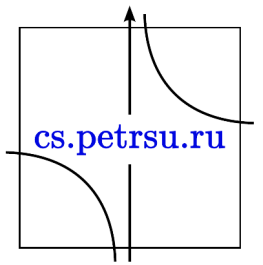
2. Анализ граничных значений

- .



3. Сравнительное тестирование

- .



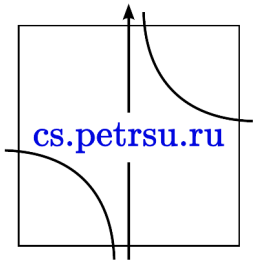
Заключение по методам тестирования

! Методы черного и белого ящиков не заменяют и взаимно дополняют друг друга.

- Для специальных задач существуют дополнительные методы
  1. Тестирование UI
  2. Тестирование клиент/серверной архитектуры
  3. Тестирование документации и системы помощи
  4. …

- Тестирование никогда не заканчивается, эта задача лишь переходит от разработчиков к заказчику.

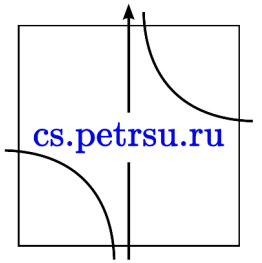
  Любой запуск ПО – это тест



§5 Организация тестирования (стратегия)

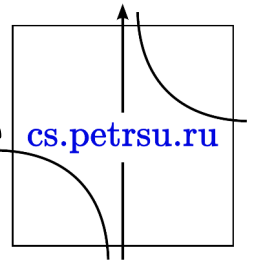
Рекомендуемый шаблон – V-Model

1. Тестирование начинается на уровне блоков/компонент, затем продолжается при интеграции компонент и завершается тестированием системы в целом
2. Различные методы тестирования могут применяться на различных фазах тестирования
3. Тестирование начинается кодировщиком ПО, а завершается независимой группой
4. Тестирование и отладка – независимые процессы, однако отладка активируется тестированием.



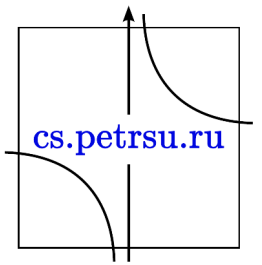
! Тестирование это неизбежная часть любой разумной разработки ПО

- Основные фазы тестирования:
  - Тестирование блоков(компонент, элементов)
  - Интеграционное тестирование (тестирование сборки)
  - Аттестационное тестирование (тестирование правильности)
  - Системное тестирование
- Критерии завершения тестирования:
  - Запланированные тесты проходят без обнаружения ошибок
  - Статистическая оценка (% ошибок/сбоев меньше заданного порога)



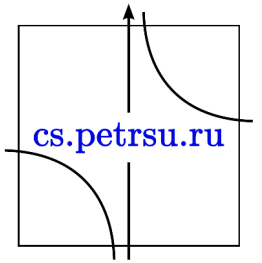
Классификация по объектам тестирования:

- Блочное (модульное) – проверка правильности работы
- Интеграционное – проверка правильности взаимосвязей
- Аттестационное – проверка требований
- Системное – проверка взаимодействия с окружением
- Регрессионное – проверка изменений



Рекомендации:

1. Формулировать требования в числовой форме (показатели)
2. Определить явно цель каждого варианта тестирования (в плане тестирования)
3. Учет специфики любого тира пользователя ПС
4. Разработка самотестируемого ПО (проверки)
5. Встроенные отладочные код
6. Предварительное инспектирование

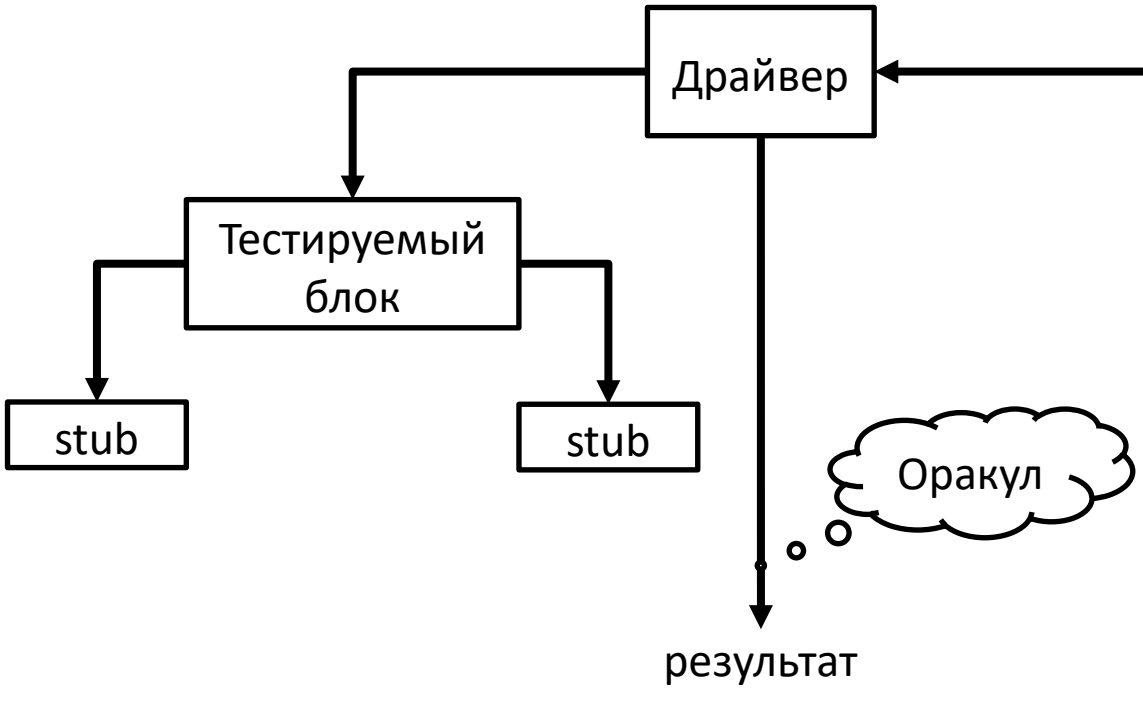


1. Тестирование блоков / Unit-тестирование

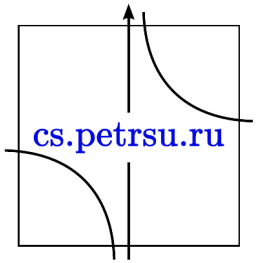
a) Требуется реализация тестируемого блока

b) Если блок делает внешний вызов, то их заменяют заглушками (stub)

c) Для вызова самого тестируемого блока используют драйвер

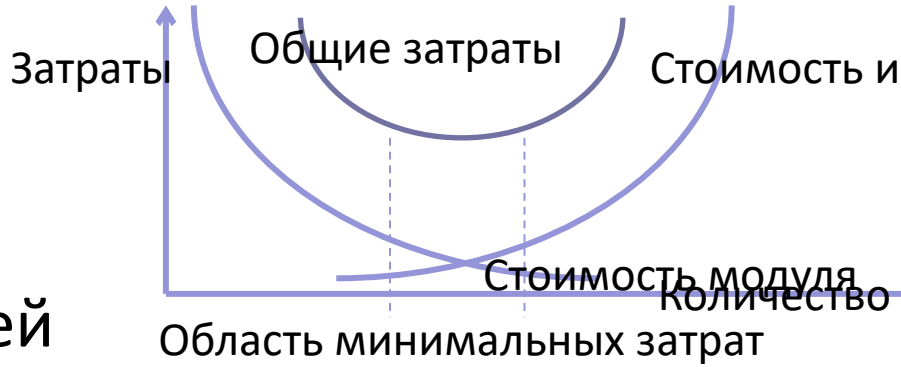


- Интерфейс
- Локальные структуры данных
- Граничные условия
- Ветви выполнения программы
- Ветви обработки ошибок



2. Интеграционное тестирование

! Если каждый модуль по отдельности работает нормально, это не означает что не будет проблем при сборке



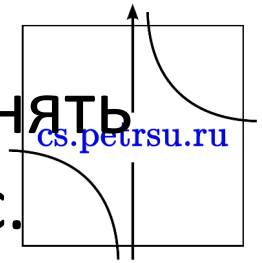
- Основная проблема –

  Интерфейс блоков/модулей

- Интеграционное тестирование – это системный подход к сборке ПО.

  Одновременно со сборкой структуры производится тестирование соответствующих интерфейсов

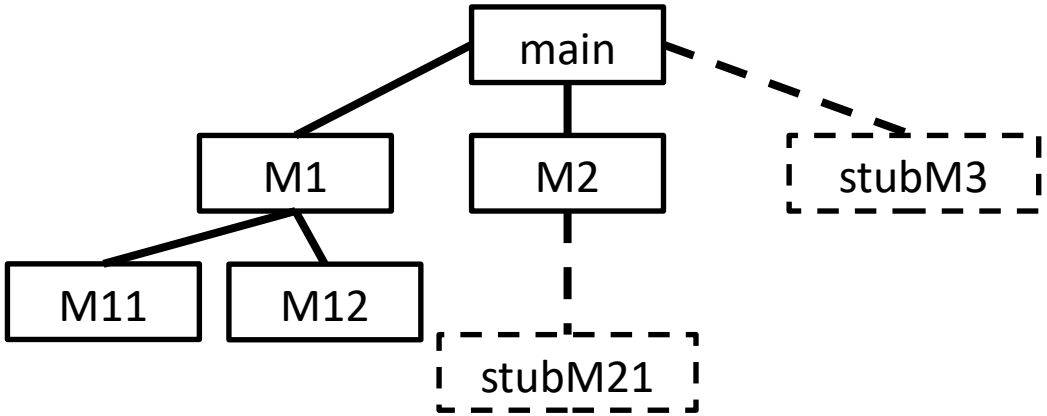
Метод “большого взрыва” не следует применять для сборки. Сборка – инкрементный процесс.



а) Нисходящее тестирование

Шаги нисходящей интеграции

1. Главный модуль используется как драйвер, все подчиненные модули заменяются заглушками
2. Одна из заглушек заменяется реальным модулем (в ширину или глубину)
3. После подключения любого модуля проводится набор тестов, проверяющих получившуюся структуру
4. Регрессионное тестирование проводится переодически



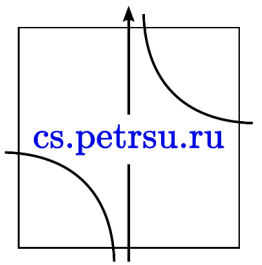
cs.petrsu.ru

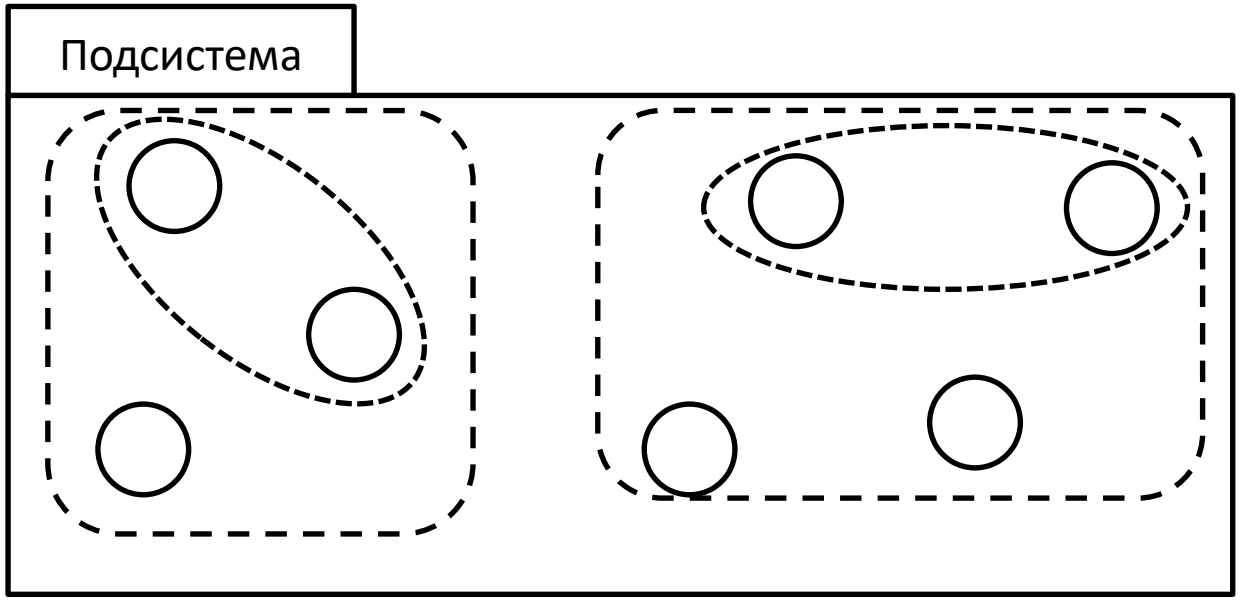
b) Восходящее тестирование

Сборка и тестирование системы начинаются с базовых модулей (низ иерархии ПС). Подчиненные модули всегда доступны и нет необходимости в заглушках

Шаги восходящей интеграции:

1. Модули нижнего уровня объединяются в кластеры, выполняющие определенную программную функцию
2. Для координации ввода/вывода тестового варианта создается драйвер кластера
3. Тестируется кластер
4. Драйвер удаляется, а кластера объединяются в структуру





Сравнение нисходящей и восходящей интеграции:

	Нисходящее	Восходящее
Недостатки	Необходимы заглушки	Система не существует как объект до окончательной сборки
Достоинства	Раннее тестирование главных управляющих функций	Отсутствуют заглушки – проще варианты тестов

3. Аттестационное тестирование

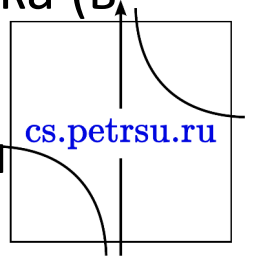
Тестирование правильности

Финальные наборы тесто – критерии аттестации системы (черный ящик)

Проверка конфигурации

- Документация разработки
- Руководство пользователя
- Листинги программ
- Установка
- …

- α-тестирование – заказчиком в организации разработчика (в лабораторных условиях)
- B-тестирование – конечный пользователь в организации заказчика без участия разработчиков

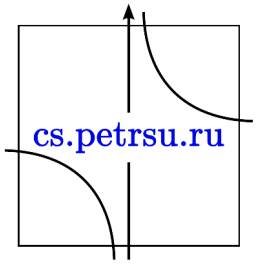


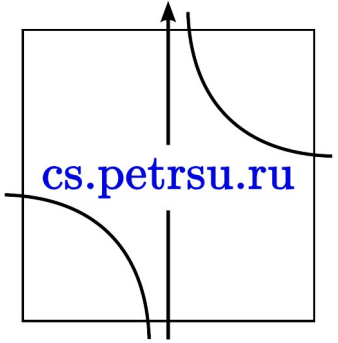
4. Системное тестирование

- Выход за рамки области действия проекта, участвуют не только разработчики
- Цель – указание причины возникновения дефекта (избежать взаимных обвинений)

В итоге: гарантировать что все системные элементы правильно объединены и выполняют назначенные функции

a) Тесты на восстановление
b) Тесты на безопасность
c) Стрессовые тесты
d) Тесты на производительность





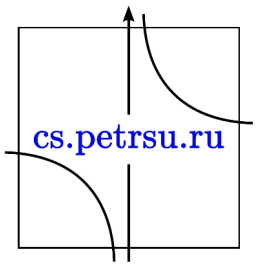
Управление проектом

Глава №7

§1 Инструментальные средства разработки и поддержки - CASE

Компоненты CASE

- Обеспечение управления проектом
  - работа с расписанием
  - распределение работ
- Управление конфигурацией ПС (среда сборки)
- Управление требованиями
- Проектирование
  - функциональное
  - объектно-ориентированное
  - варианты использования
- Инструменты отслеживания
  - переходы требования в проектные решения
  - проектных решений в программный код
- Обеспечение тестирования
- Обеспечение технической поддержки



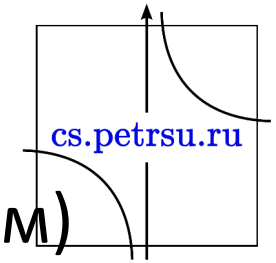
§2 Командный процесс разработки TSP – Team Software Project

Команда “запускает” каждую стадию на совещании, на котором уже известно количество предварительно определенных задач.

Предварительные мероприятия:

- Выбрать модель процесса
- Установить уровень качества
- Определить методы отслеживания уровня качества
- Определить, как команда будет принимать решения
- Определить роли и взаимодействие в команде, распределение работ

$\Rightarrow$ ПЛАН ПРОЕКТА (план управления проектом)



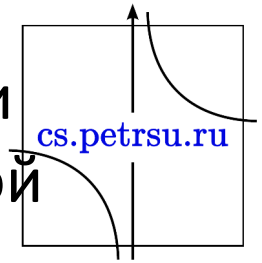
! Сочетание индивидуальной и коллективной работы

- Собрания – высокоуровневые решения
- Детальная работа происходит индивидуально или в малой команде (2-3)
- Следование заранее сформулированным правилам
- Анализ рисков проводится заранее
- Коллективная работа менее продуктивна, нежели индивидуальная, но качественней

$$W(r1 + r2) \leq W(r1) + W(r2)$$

$$Q(r1 + r2) \geq Q(r1) + Q(r2)$$

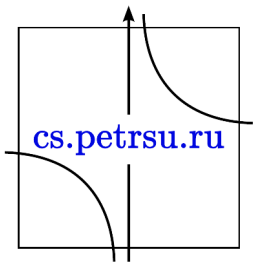
Оптимально где-то посередине – необходимо найти компромисс между коллективной и индивидуальной деятельностью



Принятие решений

- Во время проекта часто возникает перегрузка → Установка приоритетов и сортировка задач по ним
- Установка приоритетов также требует ресурсов

- … пример по приоритетам …



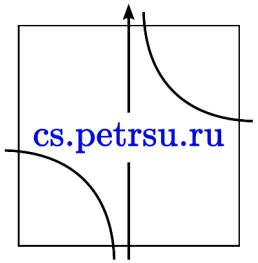
§3 Управление документацией

! Разработка ПО живет документацией

- Документация – это рабочий инструмент разработчика
- Стандарты документации

  Любая крупная компания (разработчик или заказчик) имеет свой стандарт

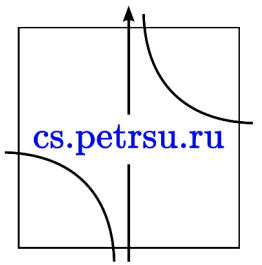
  – Такой подход облегчает:
    - Сравнение проектов
    - Переключение между проектами
    - Систематизация и унификация опыта



Проблемы управления:

- Обновление (не забыть учесть влияние обновления на другие части документации)
  - Согласованность
  - Целостность

  } Ссылки, чтобы избегать повторов
- Управления версиями
- Возможность восстановления
- Альтернативные ветви
- Стабильные версии



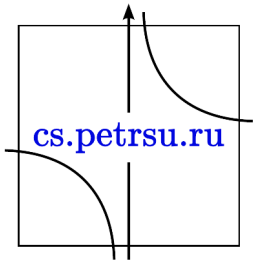
§4 Показатели ПО

Числовые показатели качества разработки: характеризуют ПО, процесс и документацию.

1. Контрольные показатели (соотносятся с требованиями и процессом разработки)
2. Прогнозируемые показатели(соотносятся с готовым продуктом)

Пример:

- Среднее значение затрат на исправление зарегистрированных неполадок
- Максимальный размер ПО а строках кода



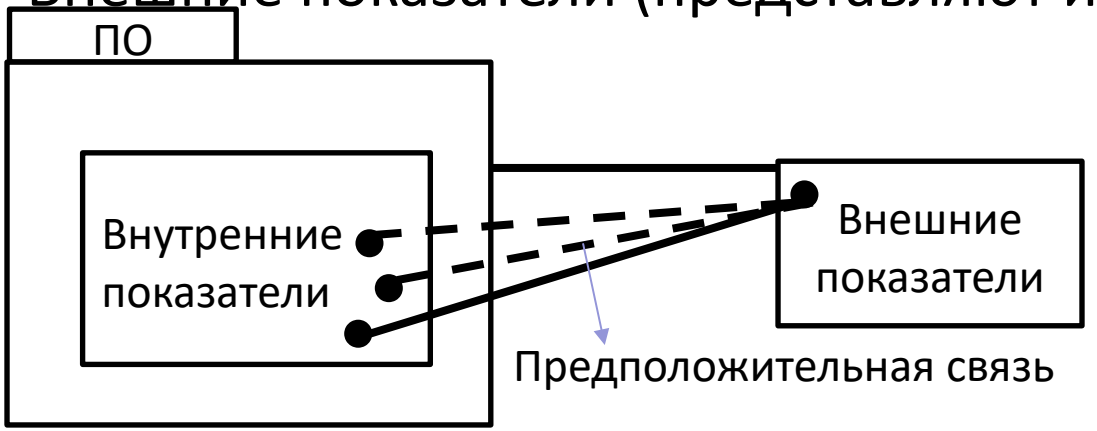
Некоторые показатели невозможно оценить напрямую

Пример:

- Удобство сопровождения, сложность кода, понятность интерфейса.

Эти показатели складываются из нескольких факторов

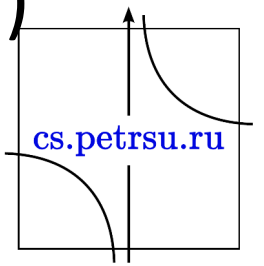
1. Внутренние показатели (можно измерить)
2. Внешние показатели (представляют интерес)



cs.petrsu.ru

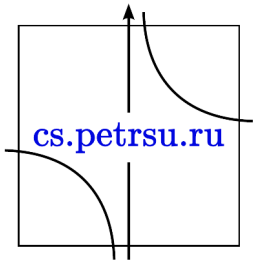
Условия оценивания внешних показателей

1. Точное и аккуратное проведение измерений внутренних показателей
2. Наличие взаимосвязи между измеренными показателями и внешними характеристиками ПО
3. Обязательное изучение этой взаимосвязи (описание в виде формулы или модели)



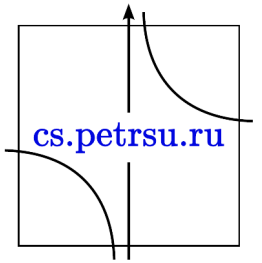
§5 Примеры показателей

1. Человеческие ресурсы
2. Взаимодействие
3. Эффективность
4. Метрики документации
5. Метрики кода
6. Метрики тестирования



§6 Модель зрелости возможностей
Модель оценки уровня развития
CMM

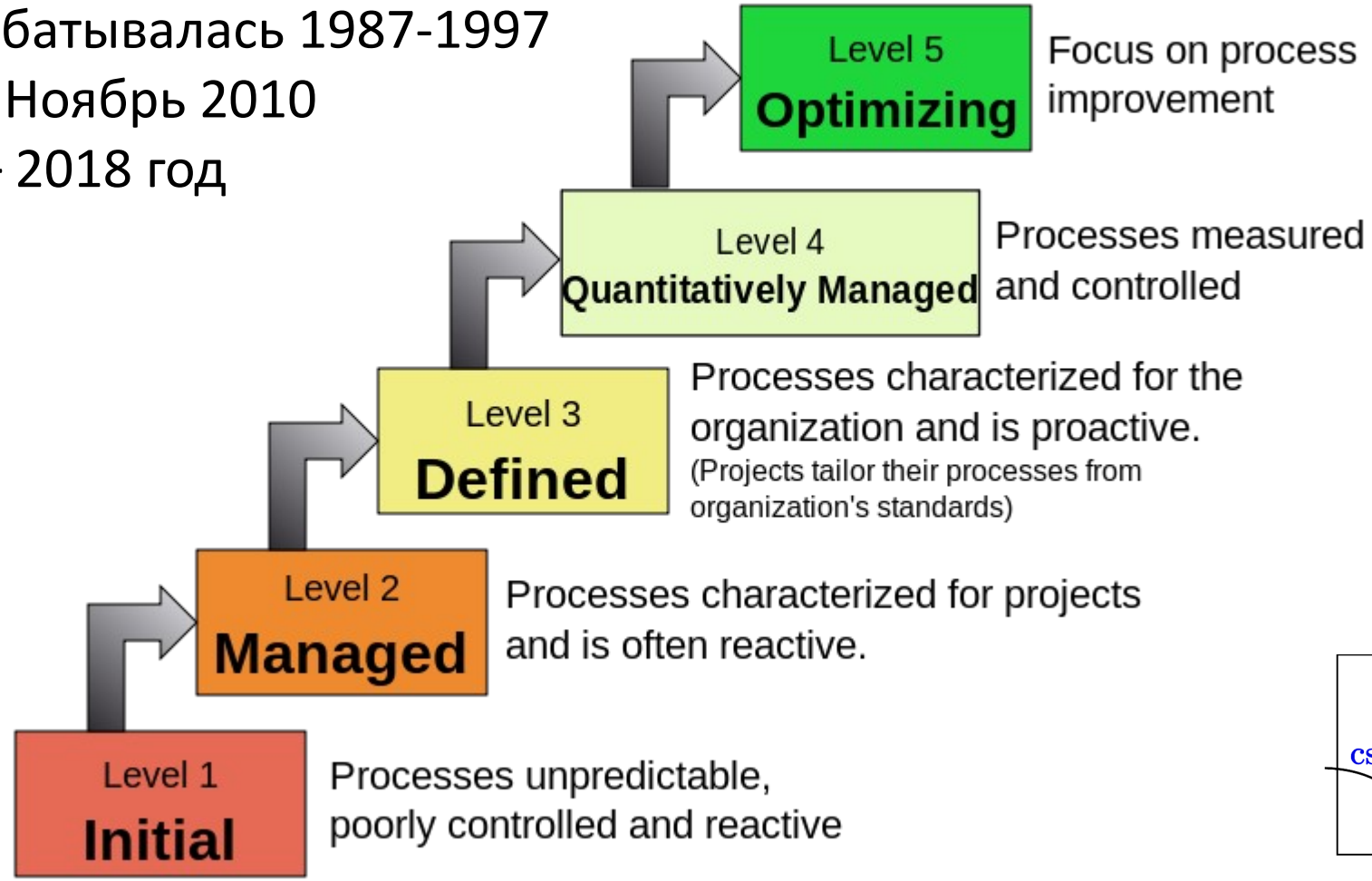
- Возможности организации по разработке качественного ПО зависят от множества факторов
- Модель классифицирует организации на 5 уровней



Capability Maturity Model Integration

Characteristics of the Maturity levels

Разрабатывалась 1987-1997
v1.3 - Ноябрь 2010
v2.0 – 2018 год



cs.petrsu.ru

Модель зрелости возможностей создания ПО
CMM - Capability Maturity Model

Начальный	Самый примитивный статус организации. Организация способна разрабатывать ПО. Организация не имеет явно осознанного процесса, и качество продукта целиком определяется индивидуальными способностями разработчиков. Один проявляет инициативу, и команда следует его указаниям. Успех одного проекта не гарантирует успех другого. При завершении проекта не фиксируются данные о трудозатратах, расписании и качестве.
Повторяемый	В некоторой степени отслеживается процесс. Делаются записи о трудозатратах и планах. Функциональность каждого проекта описана в письменной форме. В 1999-2000 гг лишь 20 % организаций имели 2-й уровень или выше.
Установленный	Имеют определённый, документированный и установленный процесс работы, не зависящий от отдельных личностей. То есть вводятся согласованные профессиональные стандарты, а разработчики их выполняют. Такие организации в состоянии достаточно надёжно предсказывать затраты на проекты, аналогичные выполненным ранее.
Управляемый	Могут точно предсказать сроки и стоимость работ. Есть база данных накопленных измерений. Но нет изменений при появлении новых технологий и парадигм.
Оптимизированный	Есть постоянно действующая процедура поиска и освоения новых и улучшенных методов и инструментов.

Спасибо за внимание