

ПМЗ

# МЕТОДЫ И СРЕДСТВА ПРОЕКТИРОВАНИЯ ПО

Лекция 3. Классификация  
технологических подходов  
(модели жизненного цикла): строгие  
(каркасные, генетические и на основе  
формальных преобразований).

## 1. Подходы со слабой формализацией


## 2 Строгие (классические, жесткие, предсказуемые) подходы

- 2.1. Каскадные технологические подходы.
  - 2.1.1. Классический каскадный подход
  - 2.1.2. Каскадно-возвратный подход
  - 2.1.3. Каскадно-итерационный подход
  - 2.1.4. Каскадный подход с перекрывающимися процессами
  - 2.1.5. Каскадный подход с подпроцессами
  - 2.1.6. Спиральная модель
- 2.2. **Каркасные подходы.**
  - 2.2.1. Рациональный унифицированный процесс (RUP)
- 2.3. Генетические подходы.
  - 2.3.1. Синтезирующее программирование
  - 2.3.2. Сборочное (расширяемое) программирование
  - 2.3.3. Конкретизирующее программирование
- 2.4. Подходы на основе формальных преобразований.
  - 2.4.1. Технология стерильного цеха
  - 2.4.2. Формальные генетические подходы

# Технологические подходы

- 3 Гибкие (адаптивные, легкие) подходы
- 3.1. Ранние технологические подходы быстрой разработки (RAD)
  - 3.1.1. Эволюционное прототипирование
  - 3.1.2. Итеративная разработка
  - 3.1.3. Постадийная разработка
- 3.2. Адаптивные подходы.
  - 3.2.1. Экстремальное программирование (XP)
  - 3.2.2. Адаптивная разработка
- 3.3. Подходы исследовательского программирования.
  - 3.3.1. Компьютерный дарвинизм





Рациональный  
унифицированный процесс

Rational Unified Process

RUP

# Проблемы разработки ПО

- Проекты по разработке ПО почти никогда не укладываются в запланированные сроки и бюджет
- Созданные в результате этого процесса программы почти никогда не оправдывают возлагавшихся на них надежд
- Только 26% проектов создания ИС заканчиваются успешно (Standish Group CHAOS Report)
- Источники и причины проблем:
  - Потребности клиентов почти всегда опережают возможности разработчиков
  - Разработчики пренебрегают присущими другим индустриям системным принципам: массовое производство, повторяемость процессов и компонентов, надежность, следование методологическим и технологическим принципам
  - Узкая специализация затрудняет взаимопонимание между участниками проекта



## 2.2. Каркасные

# ТЕХНОЛОГИЧЕСКИЕ ПОДХОДЫ

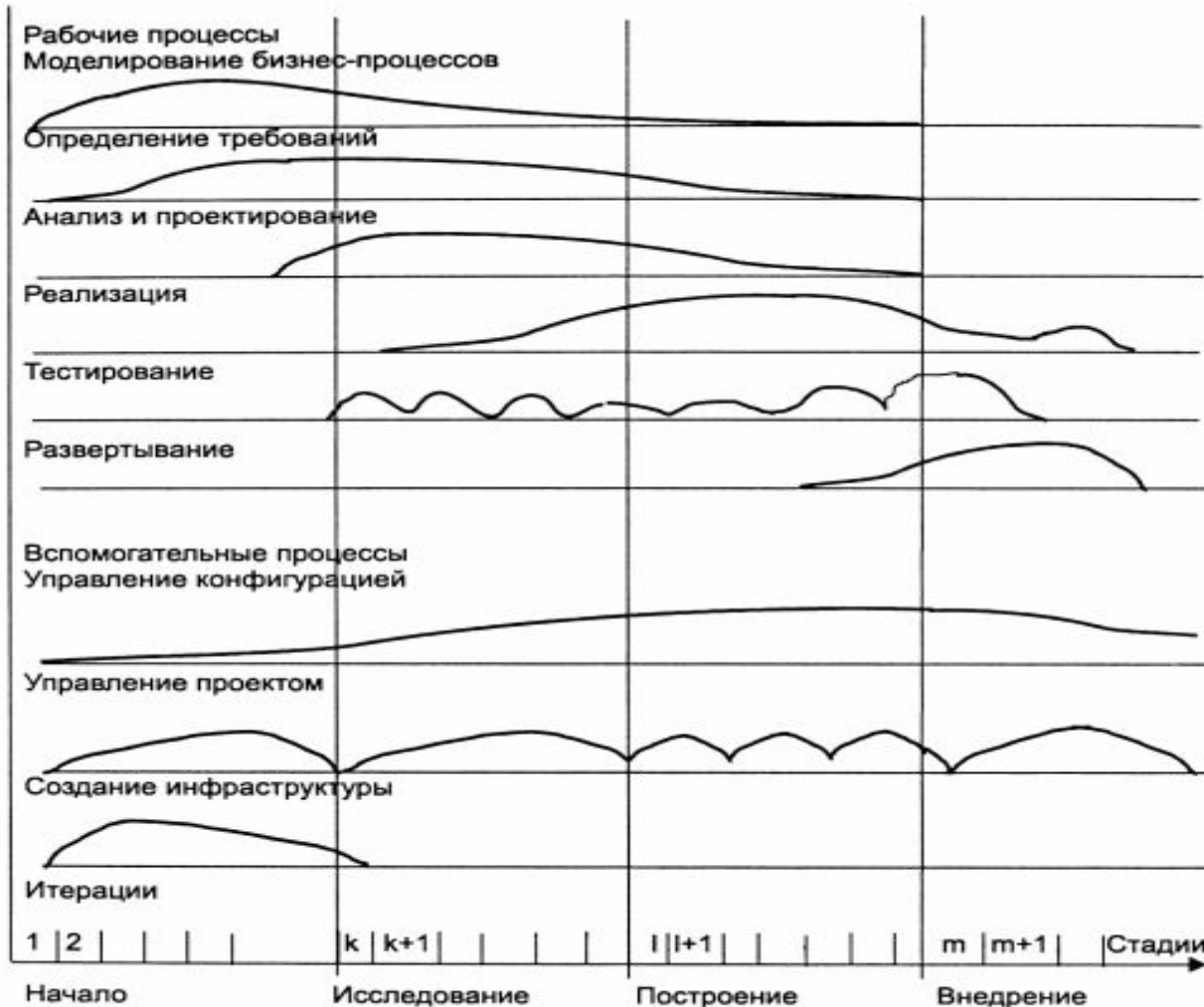
Каркасные подходы представляют собой каркас для процессов и включают их огромное количество.

- 2.2.1. Рациональный унифицированный процесс вобрал в себя лучшее из технологических подходов каскадной группы.

Процесс делится на **четыре** этапа (стадии, фазы). Каждый этап состоит из итераций. Итерация – законченный цикл разработки, вырабатывающий промежуточный продукт

В период прохождения этих фаз функционируют рабочие процессы или потоки (например, анализ и проектирование), которые состоят из ряда последовательных итераций, число итераций каждой фазы определяется индивидуально для каждого конкретного проекта. Фазы RUP нельзя отождествлять с фазами водопадной модели – их назначение и содержание принципиально различны. На рисунке они изображаются по горизонтали.

# 2.2.1. Рациональный унифицированный процесс



Основные особенности подхода:

итеративность с присущей ей гибкостью;

контроль качества; возможность выявить и устранить риски как можно раньше;

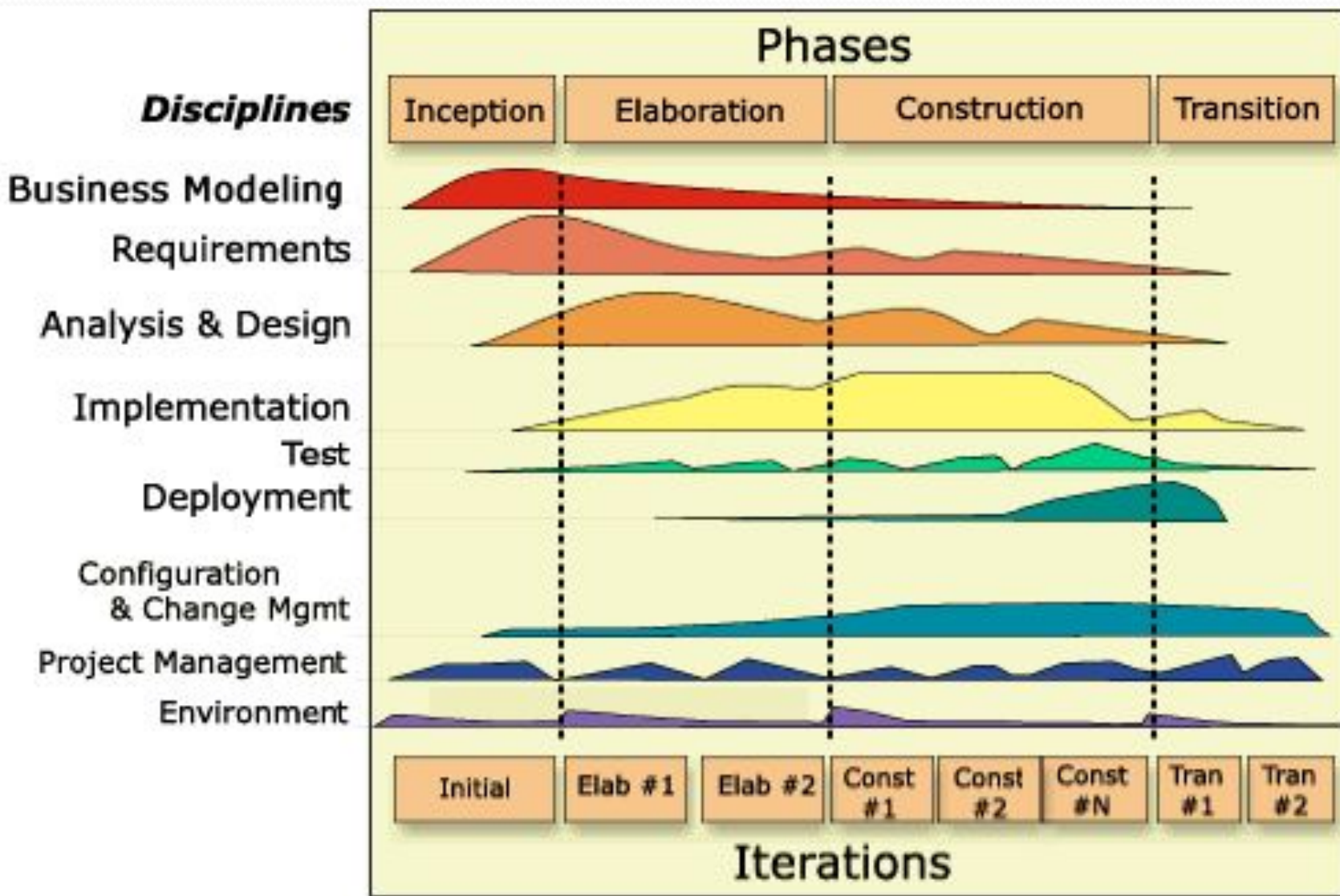
предпочтение в первую очередь отдается моделям, а не бумажным документам;

основное внимание уделяется раннему определению архитектуры;

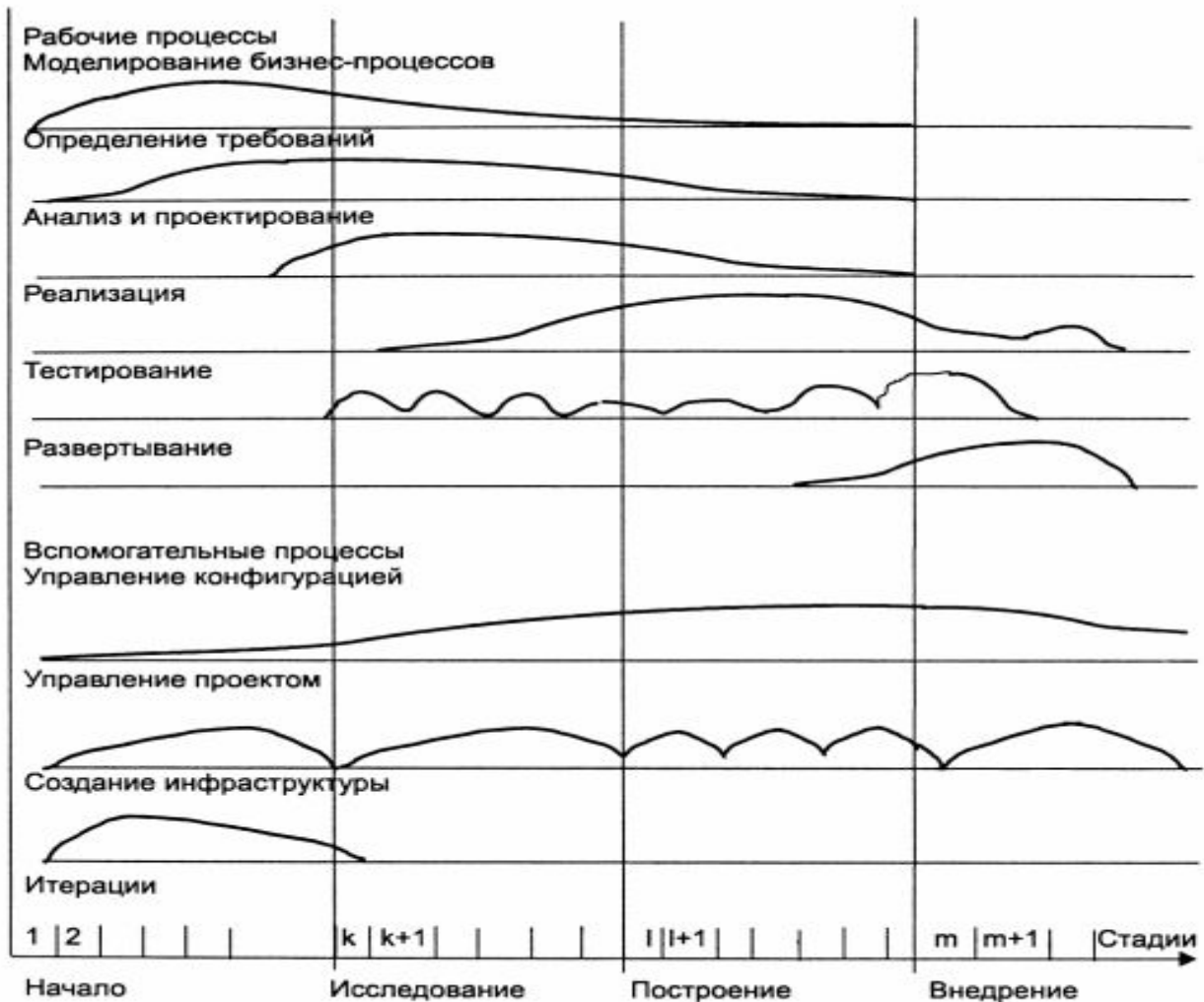
возможность конфигурирования, настройки и масштабирования.



# 2.2.1. Рациональный унифицированный процесс



# 2.2.1. Рациональный унифицированный процесс



Основные особенности подхода:

итеративность с присущей ей гибкостью;

контроль качества; возможность выявить и устранить риски как можно раньше;

предпочтение в первую очередь отдается моделям, а не бумажным документам;

основное внимание уделяется раннему определению архитектуры;

возможность конфигурирования, настройки и масштабирования.



# 2.2.1. Rational Unified Process

Авторы RUP:

- Филипп Крачтен (Philippe Kruchten),
- Грейди Буч (Grady Booch),
- Джеймс Рамбо (James Rumbaugh)
- Айвар Якобсон (Ivar Jacobson)

**Создатели нотации UML**

Грейди Буч,  
Джеймс Рамбо,  
Айвар Якобсон

Термин RUP означает как методологию разработки, так и продукт компании IBM (ранее – Rational) для управления процессами разработки. Методология RUP описывает абстрактный общий процесс, на основе которого организация или проектная команда создает специализированный процесс, ориентированный на ее потребности

- Начало разработки 1995 г.
- Первая версия 1998г.
- Наиболее глубоко проработана
- Объединяет инкрементную и эволюционную итеративные методологии
- Базируется на UML
- На всех стадиях используются программные метрики

# Фазы RUP

## Рабочие процессы RUP (Disciplines):

Бизнес-моделирование  
(Business Modeling)

Управление требованиями  
(Requirements Management)

Анализ и проектирование  
(Analysis and Design)

Реализация (Implementation)  
Тестирование (Test)

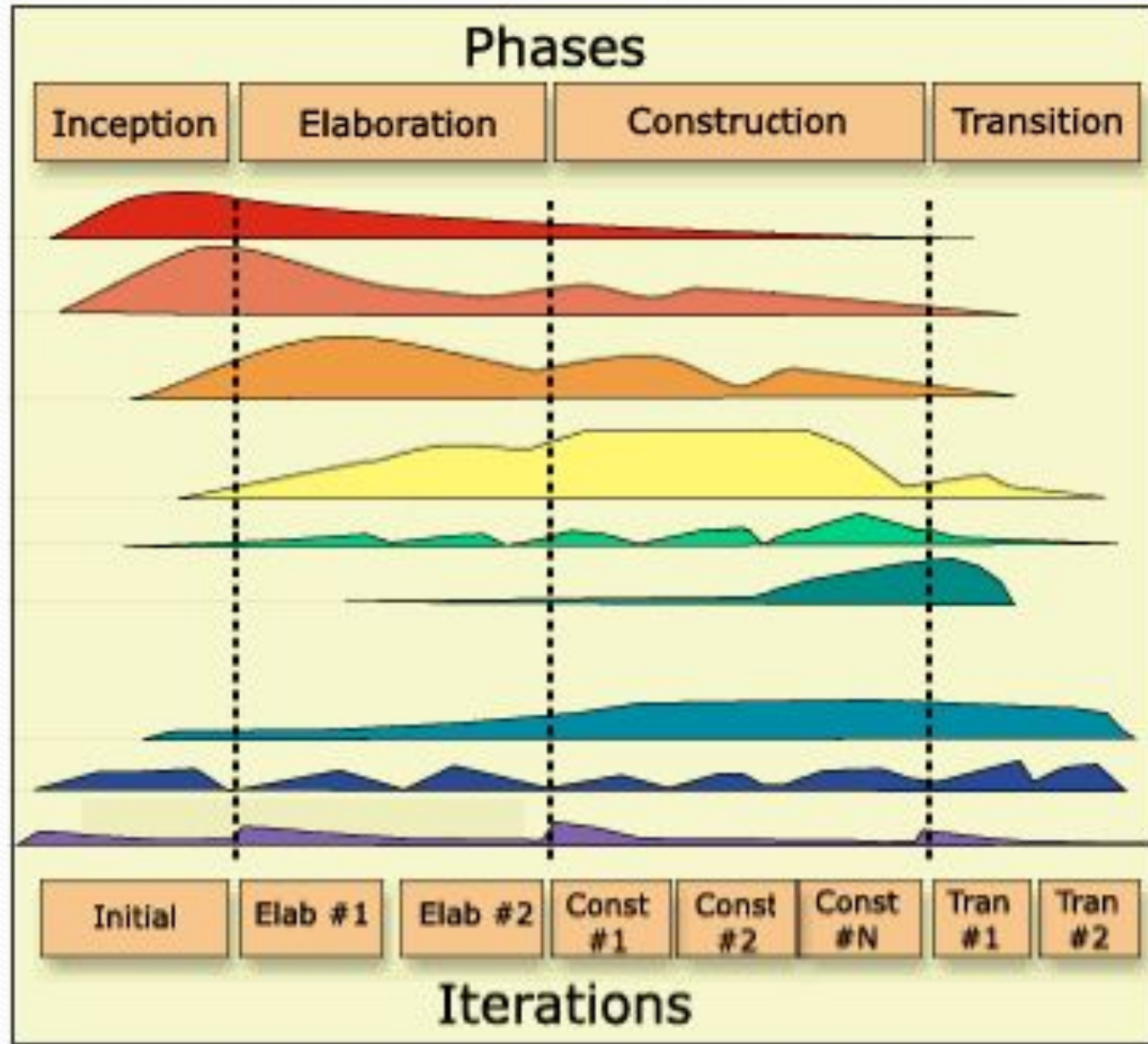
Развертывание (Deployment)

## Вспомогательные процессы:

Управление конфигурациями и  
изменениями (Configuration and  
Change Management)

Управление проектом (Project  
Management)

Среда (Environment)







# Основные характеристики процесса RUP

# Разработка требований в

## RUP

- Для описания требований в RUP используются прецеденты использования (use cases). Айвар Якобсон (один из создателей RUP) является также автором концепции прецедента использования. Полный набор прецедентов использования системы вместе с логическими отношениями между ними (прецеденты могут включать и расширять другие прецеденты) называется моделью прецедентов использования.
- Каждый прецедент использования – это описание сценария взаимодействия пользователя с системой, полностью выполняющего конкретную пользовательскую задачу. Разумеется, не имеет смысла документировать в виде прецедентов нефункциональные требования (к производительности, качеству т.д.). Однако согласно RUP все функциональные требования должны быть представлены в виде прецедентов использования. Считается, что модель прецедентов дает более целостное представление о функциональности системы по сравнению с традиционным описанием требований (перечислением функций, которыми должна обладать система).



# Итеративная разработка в RUP

- Проект в RUP состоит из последовательности итераций с рекомендованной продолжительностью от 2 до 6 недель. Основной единицей планирования итераций является прецедент использования. Перед началом очередной итерации определяется набор прецедентов использования, которые будут реализованы к ее завершению.
- Итеративная модель позволяет вносить необходимые изменения в требования, проектные решения и реализацию в ходе проекта.

# Архитектура в RUP

- RUP – ориентированная на архитектуру методология. Считается, что реализация и тестирование архитектуры системы должны начинаться на самых ранних стадиях проекта. RUP использует понятие исполняемой архитектуры (executable architecture) – основы приложения, позволяющей реализовать архитектурно значимые прецеденты использования. Основы исполняемой архитектуры должны быть реализованы как можно раньше. Это позволяет оценить адекватность принятых архитектурных решений и внести необходимые коррективы еще в начале проекта. Таким образом, для первых нескольких итераций необходимо выбирать прецеденты, которые требуют реализации большей части архитектурных компонентов.
- RUP поощряет использование визуальных средств для анализа и проектирования. Как правило, используется нотация и, соответственно, средства моделирования UML (такие как Rational Rose). Модель предметной области документируется в виде диаграммы классов, модель прецедентов использования – при помощи диаграммы прецедентов, взаимодействие компонентов системы между собой описывается диаграммой последовательности и т.д.



# Жизненный цикл проекта в RUP

Состоит из четырех фаз. Последовательность этих фаз фиксирована, но число итераций, необходимых для завершения каждой фазы, определяется индивидуально для каждого конкретного проекта. Фазы RUP нельзя отождествлять с фазами водопадной модели – их назначение и содержание принципиально различны.

**Начало (Inception).** Фаза Начало обычно состоит из одной итерации. В ходе выполнения этой фазы необходимо:

- Определить видение и границы проекта.
- Создать экономическое обоснование (business case).
- Идентифицировать большую часть прецедентов использования и подробно описать несколько ключевых прецедентов.
- Найти хотя бы одно возможное архитектурное решение.
- Оценить бюджет, график и риски проекта.

Если после завершения первой итерации заинтересованные лица приходят к выводу о целесообразности выполнения проекта, проект переходит в следующую фазу. В противном случае проект может быть отменен или проведена еще одна итерация фазы Начало.

**Проектирование (Elaboration).** В результате выполнения этой фазы на основе требований и рисков проекта создается основа архитектуры системы. Проектирование может занимать до двух-трех итераций или быть полностью пропущенным (если в проекте используется архитектура существующей системы без изменений). Целями этой фазы являются:

- Детальное описание большей части прецедентов использования.
- Создание оттестированной (при помощи архитектурно значимых прецедентов использования) базовой архитектуры.
- Снижение основных рисков и уточнение бюджета и графика проекта.

В отличие от модели водопада, основным результатом этой фазы является не множество документов со спецификациями, а действующая система с 20-30% реализованных прецедентов использования.

**Построение (Construction).** В этой фазе (длящейся от двух до четырех итераций) происходит разработка окончательного продукта. Во время ее выполнения создается основная часть исходного кода системы и выпускаются промежуточные демонстрационные прототипы.

## **Внедрение (Transition)**

Целями фазы Внедрения являются проведение бета-тестирования и тренингов пользователей, исправление обнаруженных дефектов, развертывание системы на рабочей площадке, при необходимости – миграция данных. Кроме того, на этой фазе выполняются задачи, необходимые для проведения маркетинга и продаж.

Фаза внедрения занимает от одной до трех итераций. После ее завершения проводится анализ результатов выполнения всего проекта: что можно изменить для улучшения эффективности в будущих проектах?

# Рабочий процесс RUP

В RUP определены шесть инженерных дисциплин. В них входят:

- Бизнес-моделирование (Business Modeling) – исследование и описание существующих бизнес-процессов заказчика, а также поиск их возможных улучшений.
- Управление требованиями (Requirements Management) – определение границ проекта, разработка функционального дизайна будущей системы и его согласование с заказчиком.
- Анализ и проектирование (Analysis and Design) – проектирование архитектуры системы на основе функциональных требований и ее развитие на протяжении всего проекта.
  - Создание статического и динамического представления системы
- Реализация (Implementation) – разработка, юнит-тестирование и интеграция компонентов системы.
  - Создание программного кода
- Тестирование (Test) – поиск и отслеживание дефектов в системе, проверка корректности реализации требований.
  - Проверка система в целом
- Развертывание (Deployment) – создание дистрибутива, установка системы, обучение пользователей.



# Практика RUP

Часто RUP считают тяжеловесным процессом с высоким уровнем формализма. Это не совсем так, поскольку процесс RUP может (и должен) быть настроен под специфику конкретной организации и проекта. Небольшие проекты требуют низкой степени формализма, крупные проекты с географически распределенной командой – высокой. С практической точки зрения, надо дать ответы на вопросы наподобие следующих:

- Какие артефакты будут использоваться проектной командой? Например, будет ли создана формальная спецификация приемочного тестирования?
- Каким образом артефакты будут документированы? Одна и та же модель может существовать в уме разработчиков, быть нарисованной на бумаге или созданной в одном из средств UML-моделирования.
- Насколько формальным будет процесс создания артефактов? Например, для каких документов обязательно проводить рецензирование и кто будет это делать?

Однако широкие возможности настройки не делают внедрение RUP простым. Эффективному использованию RUP препятствует множество сложностей. Часто последовательность фаз

- начало-проектирование-построение-внедрение ошибочно интерпретируют как фазы
- анализ-дизайн-реализация-внедрение из водопадной модели – это практически сводит на нет преимущества RUP.
- Для большинства людей оказывается непривычной концепция документирования требований в виде прецедентов использования – трудности возникают как с написанием, так и с пониманием прецедентов. В результате спецификация требований, созданная проектной командой, может оказаться совершенно невнятным документом, который никто не будет читать.
- Для полноценного внедрения RUP организация должна затратить значительные средства на обучение сотрудников. При этом попытка обойтись своими силами скорее всего будет обречена на неудачу – необходимо искать специалиста по процессам (process engineer) с соответствующим опытом или привлекать консультантов.

# 2.2.1. Rational Unified Process

Методология разработки программного обеспечения, созданная компанией Rational Software. В основе методологии лежат 6 основных принципов:–

- компонентная архитектура, реализуемая и тестируемая на ранних стадиях проекта;
- работа над проектом в сплочённой команде, ключевая роль в которой принадлежит архитекторам;
- ранняя идентификация и непрерывное устранение возможных рисков;
- концентрация на выполнении требований заказчиков к исполняемой программе;
- ожидание изменений в требованиях, проектных решениях и реализации в процессе разработки;
- постоянное обеспечение качества на всех этапах разработки проекта.

Использование методологии RUP направлено на итеративную модель разработки. Особенность методологии состоит в том, что степень формализации может меняться в зависимости от потребностей проекта. Можно по окончании каждого этапа и каждой итерации создавать все требуемые документы и достигнуть максимального уровня формализации, а можно создавать только необходимые для работы документы, вплоть до полного их отсутствия. За счет такого подхода к формализации процессов методология является достаточно гибкой и широко популярной.

Методология применима:

как в небольших и быстрых проектах, где за счет отсутствия формализации требуется сократить время выполнения проекта и расходы,

так и в больших и сложных проектах, где требуется высокий уровень формализма, например, с целью дальнейшей сертификации продукта.

Это преимущество дает возможность использовать одну и ту же команду разработчиков для реализации различных по объему и требованиям.



# RUP

В терминах RUP участники проектной команды создают так называемые артефакты (work products), выполняя задачи (tasks) в рамках определенных ролей (roles). К артефактам, создаваемым на разных этапах, предъявляются жесткие требования.

Задачи разделяются по девяти процессным областям, называемым дисциплинами (discipline). В RUP определены шесть инженерных и три вспомогательные дисциплины.

# Примеры артефактов RUP

- словари,
- план разработки ПО,
- разные аспекты, связанные с рисками (например, список рисков текущего этапа),
- описание архитектуры,
- спецификации,
- модели, используемые на разных этапах проектирования,
- исходный код
- и др.



# Рабочий процесс RUP

В RUP определены шесть ИНЖЕНЕРНЫХ ДИСЦИПЛИН. В них входят:

- **Бизнес-моделирование** (Business Modeling) – исследование и описание существующих бизнес-процессов заказчика, а также поиск их возможных улучшений.
- **Управление требованиями** (Requirements Management) – определение границ проекта, разработка функционального дизайна будущей системы и его согласование с заказчиком.
- **Анализ и проектирование** (Analysis and Design) – проектирование архитектуры системы на основе функциональных требований и ее развитие на протяжении всего проекта.
  - Создание статического и динамического представления системы
- **Реализация** (Implementation) – разработка, юнит-тестирование и интеграция компонентов системы.
  - Создание программного кода
- **Тестирование** (Test) – поиск и отслеживание дефектов в системе, проверка корректности реализации требований.
  - Проверка система в целом
- **Развертывание** (Deployment) – создание дистрибутива, установка системы, обучение пользователей.

# Вспомогательные дисциплины RUP

- **Управление конфигурациями и изменениями (Configuration and Change Management)** – управление версиями исходного кода и документации, процесс обработки запросов на изменение (change requests).
- **Управление проектом (Project Management)** – создание проектной команды, планирование фаз и итераций, управление бюджетом и рисками.
- **Среда (Environment)** – создание инфраструктуры для выполнения проекта, включая организацию и настройку процесса разработки.



# RUP. Начальная стадия (Inception)

- Назначение

Запуск проекта

- Цели

- Определение области применения
- Определение элементов Use Case, критических для системы
- Определение общих черт архитектуры
- Определение общей стоимости и плана проекта
- Идентификация основных элементов риска (определение потенциальных моментов при проектировании, которые могут привести к неуспешному завершению проекта)

# RUP. Начальная стадия. Действия

- **Формулировка области применения проекта**
  - Выявление требований (функциональных требований) и ограничений (нефункциональных требований)
- **Планирование**
  - Подготовка бизнес-варианта и альтернатив развития для управления риском
  - Определение персонала
  - Определение проектного плана
  - Определение зависимости между стоимостью, планированием (время) и полезностью (функциональность)
- **Синтез предварительной архитектуры ПС**
  - Развитие решений проектирования
  - Определения используемых компонентов (разработка, покупка, повторное использование)



# RUP. Начальная стадия. Артефакты

- Спецификация **основных** проектных требований
- Начальная модель Use Case (20%)
- Начальный словарь проекта (для больших проектов из специфической предметной области важно!) на разделяемом ресурсе (файле, викихранилище)
- Начальный бизнес-вариант (развития проекта с точки зрения бизнеса, если риски не сработают)
- Начальная оценка риска (в виде списка и он приоритизируется)
- Укрупненный проектный план с этапами и итерациями

# RUP. Уточнение (Elaboration).

- Назначение
  - Создать архитектурный базис
- Цели
  - Определяются оставшиеся требования
  - Функциональные требования выражаются с помощью Use Case
- Определение архитектурной платформы системы (клиент-сервер, трехзвенную и др.)
- Отслеживание рисков, устранение наибольших рисков
- Разработка плана итераций этапа «Конструирование»



# RUP. Уточнение. Действия

- Развитие спецификации
- Формирование критических элементов Use Case, задающих дальнейшие решение
- Развитие архитектуры, выделение её компонентов

# RUP. Уточнение. Артефакты

- Модель Use Case (80%)
- Дополнительные (в том числе нефункциональные) требования
- Описание программной архитектуры
- Действующий архитектурный макет (прототип приложения, реализующий взаимодействия между основными архитектурными компонентами, или иначе: каркас, который основную функцию не выполняет, но основные взаимодействия между компонентами реализует)
- Переработанный список элементов рисков и бизнес-вариант
- План разработки всего проекта, включающий все итерации и критерии развития для каждой итерации



# RUP. Построение (Construction).

- Назначение
  - Создание ПП с начальной функциональностью
- Цели
  - Минимальная стоимость разработки
  - Быстрое получение требуемого качества
  - Быстрое получение версий

# RUP. Построение. Действия

- Управление ресурсами, контроль ресурсов (люди, используемое ПО, аппаратура)
- Оптимизация процессов разработки
- Полная разработка компонентов и их тестирование
- Оценивание реализаций продукта и корректировка списка рисков (неправильно выбранной архитектуры, неправильного построения проектной команды, неправильно купленного компонента, который не реализует требуемую задачу)



# RUP. Построение. Артефакты

- ПП, пригодный для отчуждения от разработчиков (альфа-, бета-версии и т.п.)
- Описание текущей реализации
- Руководство пользователя

# RUP. Внедрение (Transition)

- Назначение
  - Отдать ПП пользователю
  - Завершить цикл выпуска ПП
- Действия в каждой итерации
  - Выпуск версии или релизов
  - Исправление найденных в процессе бета-тестирования ошибок
- Результат
  - Законченный ПП



# RUP. Выводы

- Наиболее продуманная методология
- Подходит для больших и очень больших проектов (реже средних)
- Требуется высокой квалификации участников

## 1. Подходы со слабой формализацией

### 2 Строгие (классические, жесткие, предсказуемые) подходы

- 2.1. Каскадные технологические подходы.
  - 2.1.1. Классический каскадный подход
  - 2.1.2. Каскадно-возвратный подход
  - 2.1.3. Каскадно-итерационный подход
  - 2.1.4. Каскадный подход с перекрывающимися процессами
  - 2.1.5. Каскадный подход с подпроцессами
  - 2.1.6. Спиральная модель
- 2.2. Каркасные подходы.
  - 2.2.1. Рациональный унифицированный процесс (RUP)
- 2.3. **Генетические подходы.**
  - 2.3.1. Синтезирующее программирование
  - 2.3.2. Сборочное (расширяемое) программирование
  - 2.3.3. Конкретизирующее программирование
- 2.4. Подходы на основе формальных преобразований.
  - 2.4.1. Технология стерильного цеха
  - 2.4.2. Формальные генетические подходы

### 3 Гибкие (адаптивные, легкие) подходы

- 3.1. Ранние технологические подходы быстрой разработки (RAD)
  - 3.1.1. Эволюционное прототипирование
  - 3.1.2. Итеративная разработка
  - 3.1.3. Постадийная разработка
- 3.2. Адаптивные подходы.
  - 3.2.1. Экстремальное программирование (XP)
  - 3.2.2. Адаптивная разработка
- 3.3. Подходы исследовательского программирования.
  - 3.3.1. Компьютерный дарвинизм



## 2.3. Генетические технологические подходы

Название этой группы подходов дано Поттосином, в которой термин "генетический" связывается:

- с происхождением программы
- и дисциплиной ее создания.

## 2.3.1. Синтезирующее программирование

Синтезирующее программирование предполагает синтез программы по ее спецификации. В отличие от программы, которая написана на алгоритмическом языке и предназначена для исполнения на вычислительной машине после трансляции в исполняемый код, документ на языке спецификаций является лишь базисом для последующей реализации. Для получения этой реализации необходимо решить перечисленные ниже основные задачи

- Доопределить детали, которые нельзя выразить при помощи языка спецификации, но необходимые для получения исполняемого кода.
- Выбрать язык реализации и аппаратно-программную платформу для реализации.
- Зафиксировать отображение понятий языка спецификаций на язык реализации и аппаратно-программную платформу.
- Осуществить трансформацию представления (из спецификации в исполняемую программу на языке реализации).
- Отладить и протестировать исполняемую программу.
- Автоматическая генерация программ по спецификациям возможна для многих языков спецификаций, среди которых отметим UML.



## 2.3.2. Сборочное (расширяемое) программирование

Сборочное программирование предполагает, что программа собирается путем переиспользования уже известных фрагментов



### программирование

Сборка может осуществляться вручную или быть задана на некотором языке сборки, или извлечена полуавтоматическим образом из спецификации задачи.

Основные направления для создания техники сборочного программирования.

- Выработка стиля программирования, соответствующего принятым принципам модульности.
- Повышение эффективности межмодульных интерфейсов; важность аппаратной поддержки модульности.
- Ведение большой базы программных модулей; решение проблемы идентификации модулей и проверки пригодности по описанию интерфейса. Модули должны стать "программными кирпичиками", из которых строится программа.



## 2.3.2. Сборочное (расширяемое) программирование

Сборочное программирование тесно связано с методом повторного использования кода, причем как исходного, так и бинарного. Выделяют четыре разновидности технологических подходов сборочного программирования, которые в значительной степени определяются базисной методологией.

# программирование

**МОДУЛЬНОЕ** сборочное программирование. Этот подход был исторически первым и базировался на процедурах и функциях методологии структурного императивного программирования.

**ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ** сборочное программирование. Подход базируется на методологии объектно-ориентированного программирования и предполагает распространение библиотек классов в виде исходного кода или упаковку классов в динамически компоную библиотеку.

**КОМПОНЕНТНОЕ** сборочное программирование. Основные идеи подхода - распространение классов в бинарном виде и предоставление доступа к методам класса через строго определенные интерфейсы, что позволяет снять проблему несовместимости компиляторов и обеспечивает смену версий классов без перекомпиляции использующих их приложений..

**АСПЕКТНО-ОРИЕНТИРОВАННОЕ** сборочное программирование. Концепция компонента в этом случае дополняется концепцией аспекта - варианта реализации критичных по эффективности процедур. Аспектно-ориентированное сборочное программирование заключается в сборке полнофункциональных приложений из многоаспектных компонентов, инкапсулирующих различные варианты реализации.



## 2.3.3. Конкретизирующее программирование

Предполагает, что частные, специальные программы извлекаются из универсальной.

Наиболее известная технология конкретизирующего программирования - это подход с применением паттернов проектирования.

**Паттерн** (шаблон) проектирования (design pattern) - описание взаимодействия объектов и классов, адаптированных для решения общей задачи проектирования в конкретном контексте. Проектировщик, знакомый с паттернами, может сразу применять их к решению новой задачи, конкретизируя их. Паттерны проектирования упрощают повторное использование удачных проектных и архитектурных решений.

## 2.3.3. Конкретизирующее программирование

Паттерн состоит из четырех основных элементов:

- имени - однозначно описывающего проблему проектирования;
- задачи - описания того, когда следует применять паттерн для конкретизации;
- решения - абстрактного описания элементов дизайна и отношений между ними;
- результатов - следствий применения паттерна.

Дополнительно к паттернам существуют каркасы (framework) - наборы взаимодействующих классов, составляющих повторно используемый дизайн для конкретного класса программ. Каркас диктует определенную архитектуру приложения, в нем аккумулярованы проектные решения, общие для проектной области. Например, существуют каркасы, которые используются для разработки компиляторов.



## 2.4. Подходы на основе формальных преобразований

Эта группа подходов содержит максимально формальные требования к процессу создания программного обеспечения

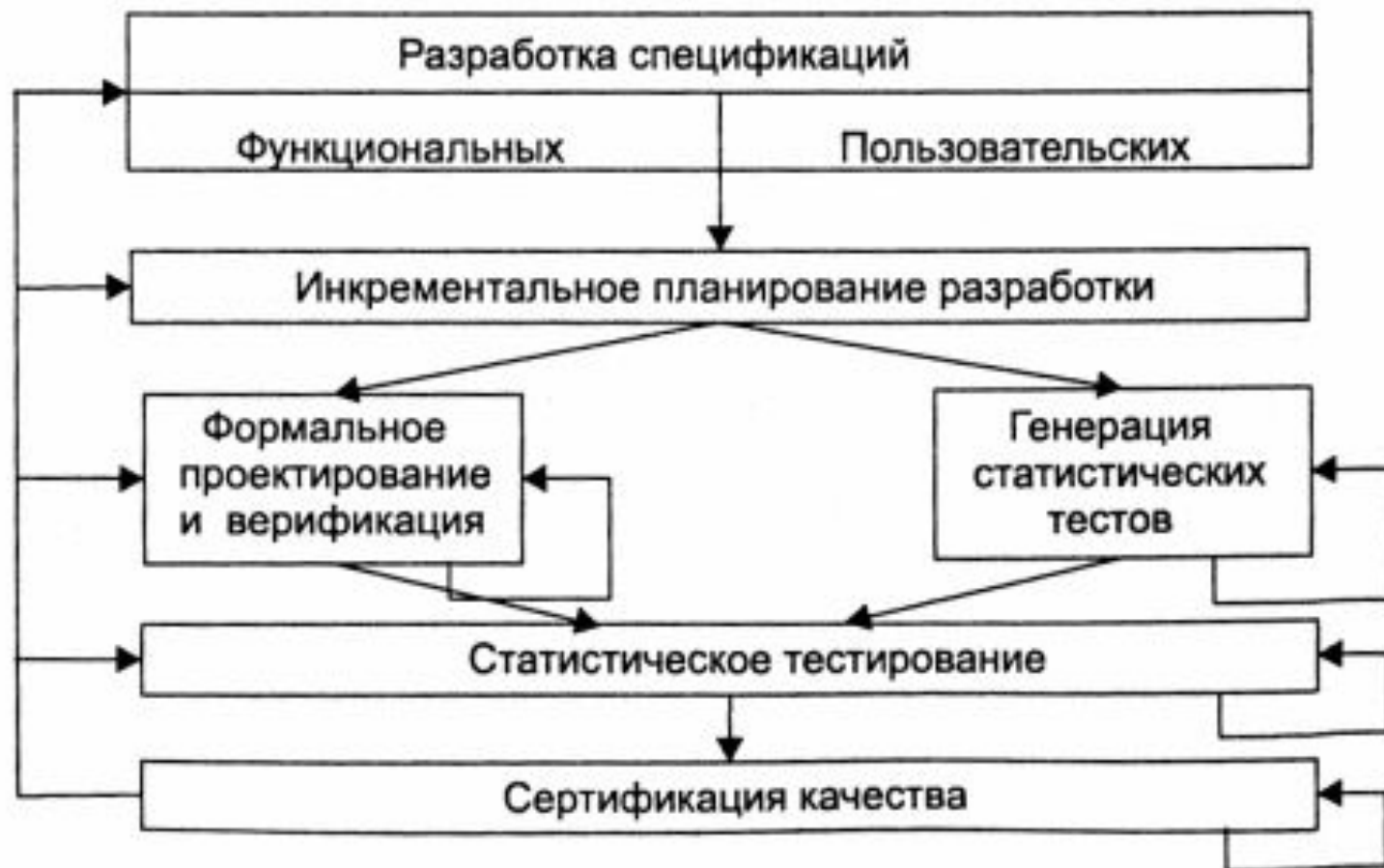
## 2.4.1. Технология стерильного цеха

Основные идеи технологии стерильного цеха (cleanroom process model) были предложены Харланом Миллзом в середине 80-х годов XX века. Технология складывается из следующих частей:

- разработка функциональных и пользовательских спецификаций;
- инкрементальное планирование разработки;
- формальная верификация;
- статистическое тестирование.



## 2.4.1. Технология стерильного цеха



## 2.4.1. Технология стерильного цеха

Процесс проектирования связан с представлением программы как функции, в виде так называемых "ящиков":

- **черного** ящика с фиксированными аргументами (стимулами) и результатами (ответами);
- ящика **с состоянием**, в котором выделяется внутреннее состояние;
- **прозрачного** (белого) ящика, представляющего реализацию в виде совокупности функций при пошаговом уточнении.

**Метод ящиков удобен, когда нам надо работать с функциями либо с закрытой, либо с открытой реализацией.**



## 2.4.1. Технология стерильного цеха

Использование ящиков определяют следующие три принципа:

- все определенные при проектировании данные скрыты (инкапсулированы) в ящиках;
- все процессы определены как использующие ящики последовательно или параллельно;
- каждый ящик занимает определенное место в системной иерархии.

## 2.4.1. Технология стерильного цеха

Черный ящик представляет собой точную спецификацию внешнего, видимого с пользовательской точки зрения поведения. Ящик получает стимулы  $S$  от пользователя и выдает ответ  $R$ . Каждый ответ черного ящика определяется его текущей историей стимулов  $SH$  как функция:

$$(S, SH) \rightarrow (R)$$



## 2.4.1. Технология стерильного цеха

На основании одних и тех же стимулов мы можем получить разные ответы, базирующиеся на истории использования.

Рассмотрим калькулятор с двумя историями стимулов:

Clear 3 1 4

Clear 3 1 4 +

Если следующим стимулом будет число 1, то на основании первой истории калькулятор выдаст ответ 3141, а на основании второй - 315.

Определять поведение черных ящиков очень удобно с помощью табличных спецификаций.

## 2.4.1. Технология стерильного цеха

Ящик с состояниями получаем из черного ящика выделением элементов истории стимулов, которые сохраняют состояние (инварианты состояний) в процессе выполнения действий в черном ящике. Функция преобразования ящика с состояниями выглядит так:

$$(S, OS) \rightarrow (R, NS)$$

Здесь OS - старое состояние, а NS - новое. Хотя с пользовательской точки зрения поведение черного ящика и ящика с состоянием выглядит одинаково, истории стимулов заменяются ссылками на старое состояние и новое состояние, требуемое преобразованием.



## 2.4.1. Технология стерильного цеха

Прозрачный ящик получаем из ящика с состояниями, определяя процедуру, выполняющую требуемое преобразование. Таким образом, прозрачный ящик - это просто программа, реализующая соответствующий ящик с состоянием.

$(S, OS) \rightarrow (R, NS)$  by procedure

Проектирование процедуры прозрачного ящика выглядит как последовательное определение функций назначения и преобразование их в структуру управления и новые функции назначения.

## 2.4.1. Технология стерильного цеха

В данной технологии отсутствует **ОТЛАДКА**. Его заменяет процесс формальной верификации. Для каждой управляющей структуры проверяется соответствующее условие корректности.

Технология стерильного цеха предполагает бригадную работу, т. е. проектирование, уточнение, инспекция и подготовка тестов ведется разными людьми.



## 2.4.2. Формальные генетические подходы

Сложились методы программирования, обладающие свойством доказательности и не теряющие это точное, накопленное знание.

Три таких метода соответствуют уже исследованным генетическим подходам, но с учетом формальных, математических спецификаций.

## 2.4.2. Формальные генетические подходы

- Формальное СИНТЕЗИРУЮЩЕЕ программирование использует математическую спецификацию - совокупность логических формул. Существуют две разновидности синтезирующего программирования:
  - ЛОГИЧЕСКОЕ, в котором программа извлекается как конструктивное доказательство из спецификации, понимаемой как теорема;
  - ТРАНСФОРМАЦИОННОЕ, в котором спецификация рассматривается как уравнение относительно программы и символическими преобразованиями превращается в программу.
- Формальное СБОРОЧНОЕ программирование использует спецификацию как композицию уже известных фрагментов.
- Формальное КОНКРЕТИЗИРУЮЩЕЕ программирование использует такие подходы, как смешанные вычисления и конкретизацию по аннотациям.



## 2.4.2. Формальные генетические подходы

Одной из наиболее интересных современных работ в области формальных генетических подходов является В-технология. На ее основе была осуществлена разработка системы управления парижским метрополитеном "METEOR".