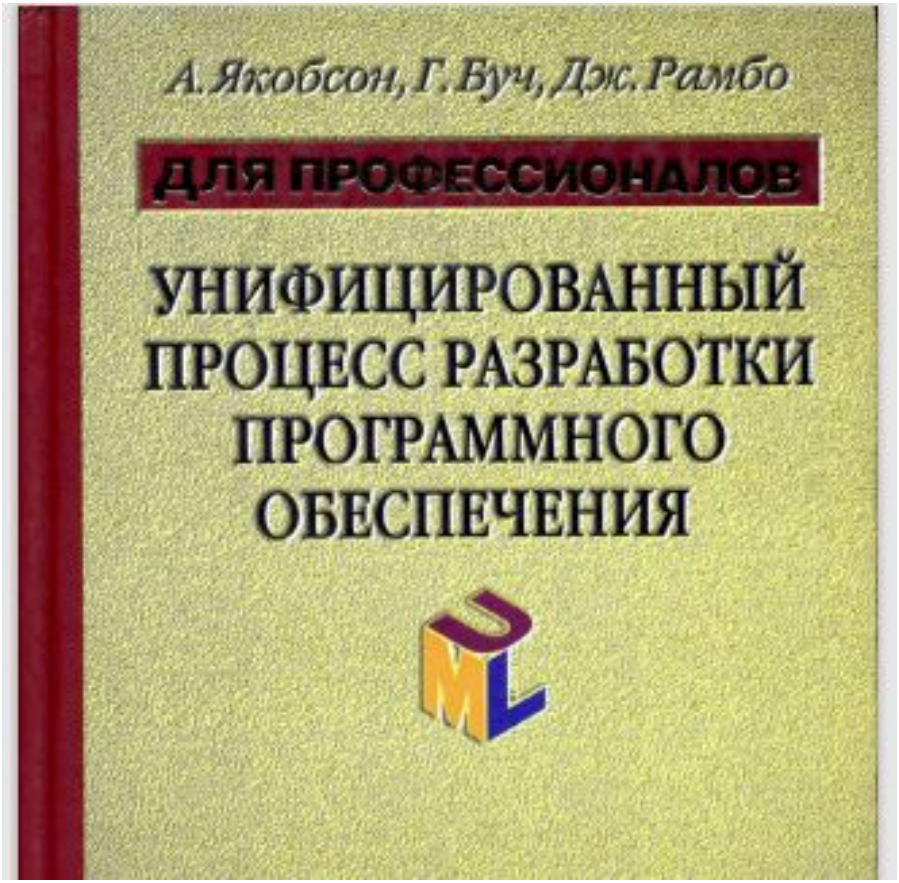
RUP (Rational Unified Process)

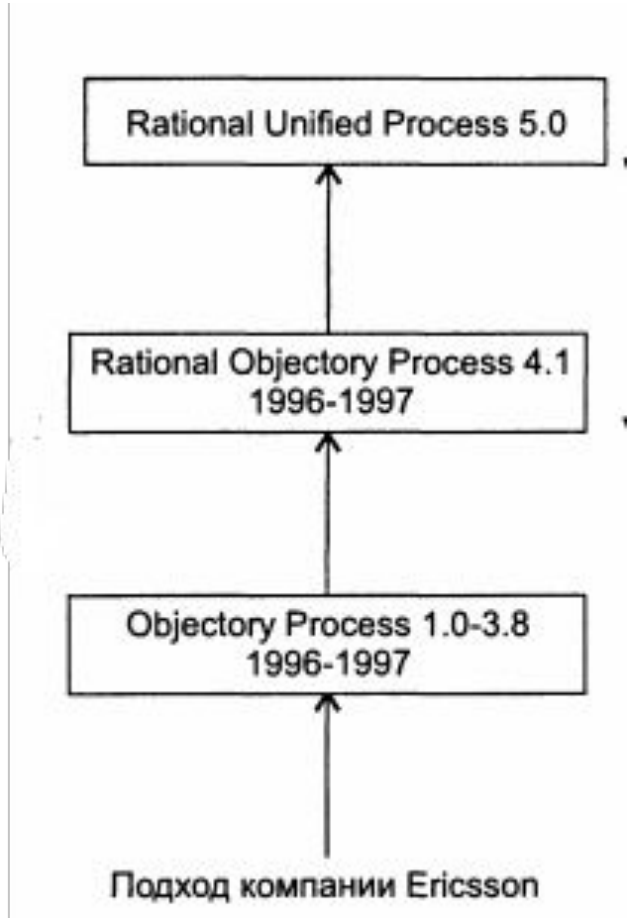


- **Унифицированный процесс Rational** — это универсальная методология распределения задач и сфер ответственности при разработке программного обеспечения. Её цель – создание высококачественного программного обеспечения, отвечающего потребностям и запросам пользователей.
- **Методология RUP** была разработана в компании **Rational Software Corporation**, которую в 2003 году купила **IBM**.

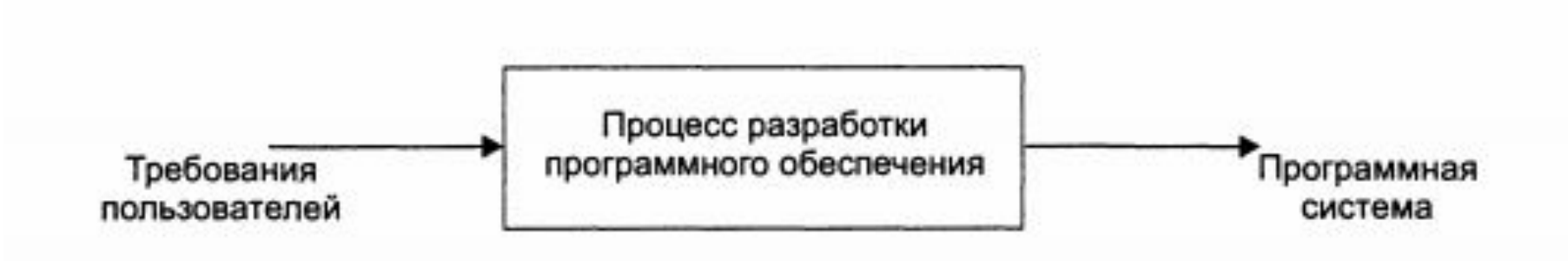
Унифицированный язык моделирования

- В 1994 году Гради Буч и Джеймс Рамбо, работавшие в компании Rational Software, объединили свои усилия для создания нового языка объектно-ориентированного моделирования. За основу языка ими были взяты методы моделирования Object-Modeling Technique и Booch. OMT был ориентирован на анализ, а Booch — на проектирование программных систем.
- Осенью 1995 года к компании Rational присоединился Ивар Якобсон, автор метода Object-Oriented Software Engineering — OOSE. OOSE обеспечивал превосходные возможности для спецификации бизнес-процессов и анализа требований при помощи сценариев использования. OOSE был также интегрирован в унифицированный метод.
- На этом этапе основная роль в организации процесса разработки UML перешла к консорциуму OMG (Object Management Group). Группа разработчиков в OMG, в которую также входили Буч, Рамбо и Якобсон («три амиго»), выпустила спецификации UML версий 0.9 и 0.91 в июне и октябре 1996 года.

Rational Unified Process создан в 1998



- Унифицированный процесс — это больше чем единичный процесс, это обобщенный каркас процесса, который может быть специализирован для широкого круга программных систем, различных областей применения, уровней компетенции и размеров проекта



Основные особенности

- процесс управляется вариантами использования
- процесс компонентно-ориентирован. Это означает, что создаваемая программная система строится на основе программных компонентов, связанных хорошо определенными интерфейсами
- процесс использует Унифицированный язык моделирования. Фактически Унифицированный язык моделирования является неотъемлемой частью Унифицированного процесса — они и разрабатывались совместно

Унифицированный процесс управляется вариантами использования

- Программная система создается для обслуживания ее пользователей. Следовательно, для построения успешной системы мы должны знать, в чем нуждаются и чего хотят ее будущие пользователи.
- Варианты использования обеспечивают функциональные требования. Сумма всех вариантов использования составляет модель вариантов использования, которая описывает полную функциональность системы. Эта модель заменяет традиционное описание функций системы. Считается, что описание функций может ответить на вопрос: что система предположительно делает? Подход вариантов использования может быть охарактеризован добавкой трех слов в конец этого вопроса: для каждого пользователя? Эти три слова представляют собой очень важное дополнение. Они побуждают нас думать в понятиях результата, значимого для пользователя, а не только в понятиях функций, которые хорошо бы иметь.

- Варианты использования — это не только средство описания требований к системе. Они также направляют ее проектирование, реализацию и тестирование, то есть **они направляют процесс разработки**. Основываясь на модели вариантов использования, разработчики создают серию моделей проектирования и реализации, которые осуществляют варианты использования. Тестеры тестируют реализацию для того, чтобы гарантировать, что компоненты модели реализации правильно выполняют варианты использования. Таким образом, варианты использования не только инициируют процесс разработки, но и служат для связи отдельных его частей.
- **Управляемый вариантами** использования означает, что процесс разработки проходит серии рабочих процессов, порожденных вариантами использования. Варианты использования определяются, варианты использования проектируются, и, в конце концов, варианты использования служат исходными данными, по которым тестеры создают тестовые примеры.

Унифицированный процесс ориентирован на архитектуру

- Понятие архитектуры программы включает в себя наиболее важные статические и динамические аспекты системы. Архитектура вырастает из требований к результату, в том виде, как их понимают пользователи и другие заинтересованные лица. Эти требования отражаются в вариантах использования.
- Архитектура — это представление всего проекта с выделением важных характеристик и затушевыванием деталей. Поскольку важность той или иной характеристики зависит, в частности, от правильности суждения, приходящей с опытом, результат построения архитектуры определяется людьми, которым поручена эта задача.

Варианты использования управляют разработкой архитектуры, а архитектура направляет реализуемые варианты использования

- Каждый продукт имеет функции и форму. Одно без другого не существует. В удачном продукте эти две стороны должны быть уравновешены. Функции соответствуют вариантам использования, а форма — архитектуре. Мы нуждаемся во взаимодействии между вариантами использования и архитектурой.
- Это вариант традиционной проблемы «курицы и яйца». С одной стороны, варианты использования должны, будучи реализованными, подойти к архитектуре. С другой стороны, архитектура должна предоставить возможности реализации любых понадобившихся сейчас и в будущем вариантов использования. Реально архитектура и варианты использования разрабатываются параллельно.

- Архитектор совершает следующие шаги:
- 1 Создает грубый набросок архитектуры, начиная с той части архитектуры, которая не связана с вариантами использования (так называемая платформа). Хотя эта часть архитектуры не зависит от вариантов использования, архитектор должен в общих чертах понять варианты использования до создания наброска архитектуры.
- 2 Далее архитектор работает с подмножеством выделенных вариантов использования, каждый из которых соответствует одной из ключевых функций разрабатываемой системы. Каждый из выбранных вариантов использования детально описывается и реализуется в понятиях подсистем («Классы группируются в подсистемы»), классов и компонентов.
- 3 После того как варианты использования описаны и полностью разработаны, большая часть архитектуры исследована. Созданная архитектура, в свою очередь, будет базой для полной разработки других вариантов использования.
- Этот процесс продолжается до тех пор, пока архитектура не будет признана стабильной.

Унифицированный процесс является итеративным и инкрементным

- Разработка коммерческих программных продуктов — это серьезное предприятие, которое может продолжаться от нескольких месяцев до года и более. Практично было бы разделить работу на небольшие куски или минипроекты. Каждый минипроект является итерацией, результатом которой будет приращение. Итерации относятся к шагам рабочих процессов, а приращение — к выполнению проекта. Для максимальной эффективности итерации должны быть управляемыми, то есть они должны выбираться и выполняться по плану. Поэтому их можно считать минипроектами.

Управляемый итеративный процесс имеет множество преимуществ.

- Управляемая итерация ограничивает финансовые риски затратами на одно приращение. Если разработчикам нужно повторить итерацию, организация теряет только усилия, затраченные на одну итерацию, а не стоимость всего продукта.
- Управляемая итерация снижает риск непоставки продукта на рынок в запланированные сроки. При раннем обнаружении соответствующего риска время, которое тратится на его нейтрализацию, вносится в план на ранних стадиях, когда сотрудники менее загружены, чем в конце планового периода. При традиционном подходе, когда серьезные проблемы впервые проявляются на этапе тестирования системы, время, требуемое для их устранения, обычно превышает время, оставшееся по плану для завершения всех работ, что почти всегда приводит к задержкам поставок.
- Управляемая итерация ускоряет темпы процесса разработки в целом, поскольку для более эффективной работы разработчиков и быстрого получения ими хороших результатов короткий и четкий план предпочтительнее длинного и вечно сдвигающегося.
- Управляемая итерация признает часто отвергаемый факт — что желания пользователей и связанные с ними требования не могут быть определены в начале разработки. Они обычно уточняются в последовательных итерациях. Такой образ действий облегчает адаптацию к изменениям требований.

Серьезные риски при итеративной разработке определяются и уменьшаются раньше, чем при водопадной

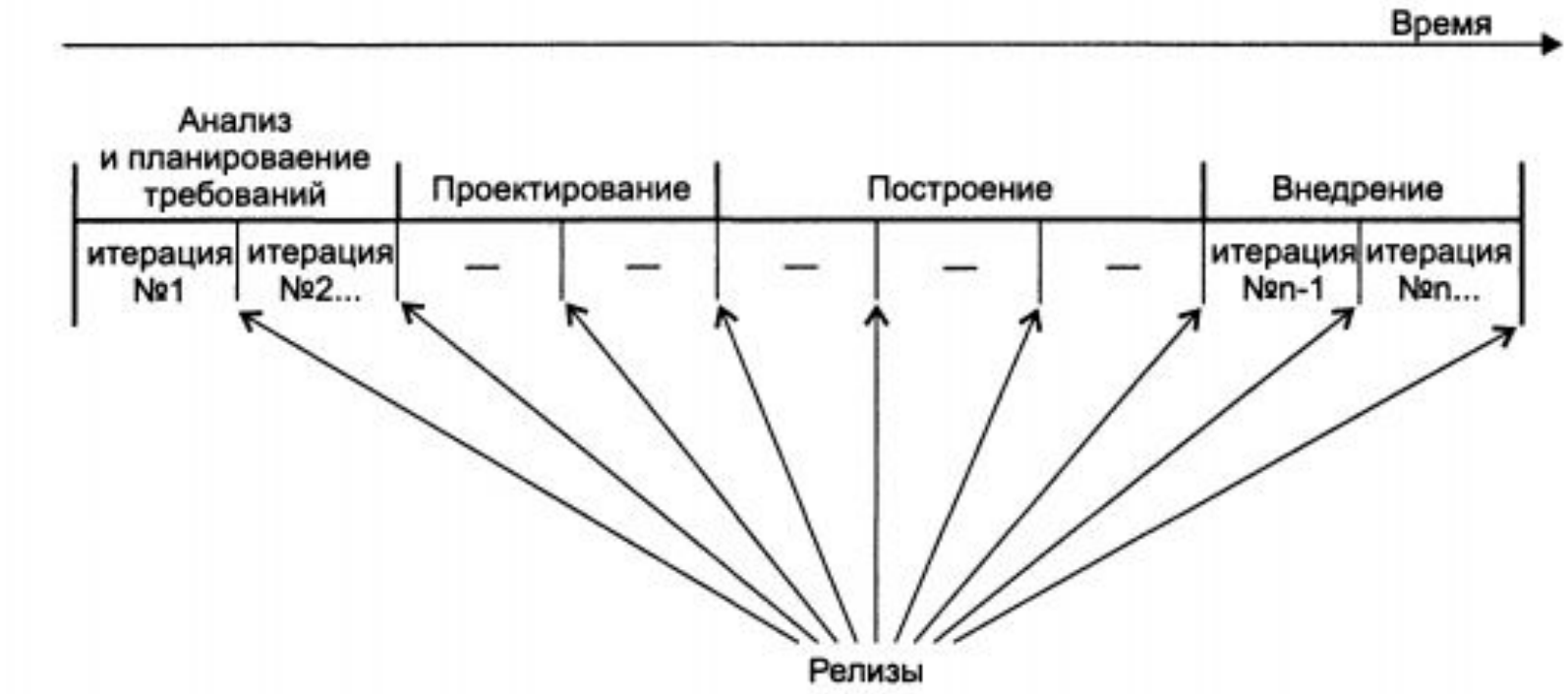
Жизненный цикл Унифицированного процесса

- Унифицированный процесс циклически повторяется. Эта последовательность повторений Унифицированного процесса, как показано на рис., представляет собой жизненный цикл системы. Каждый цикл завершается поставкой выпуска продукта заказчикам.

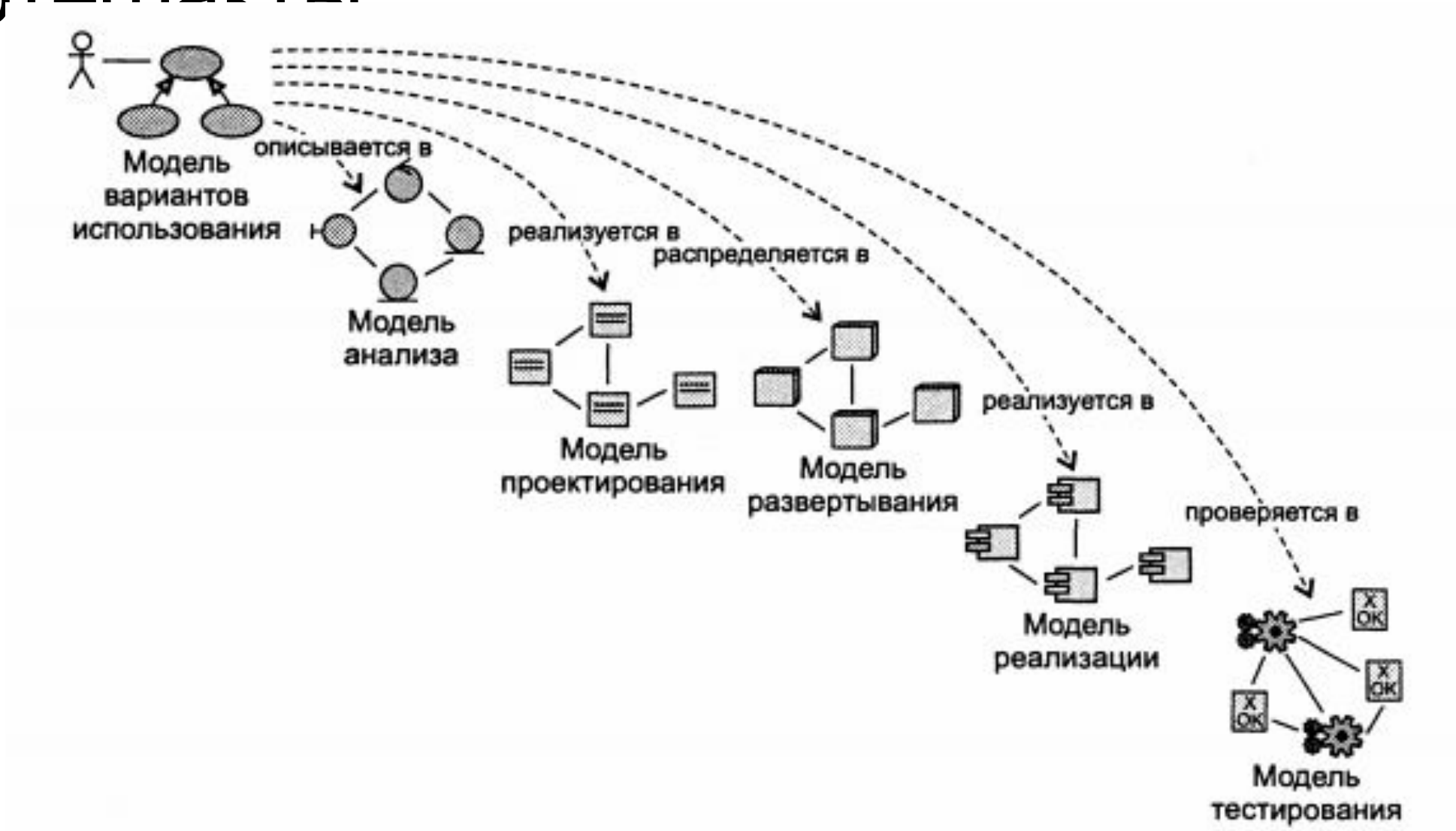
Жизнь процесса, состоящая из циклов, от его рождения до смерти

Цикл с его фазами и итерациями

- Каждый цикл состоит из четырех фаз — анализа и планирования требований, проектирования, построения и внедрения. Каждая фаза, далее подразделяется на и

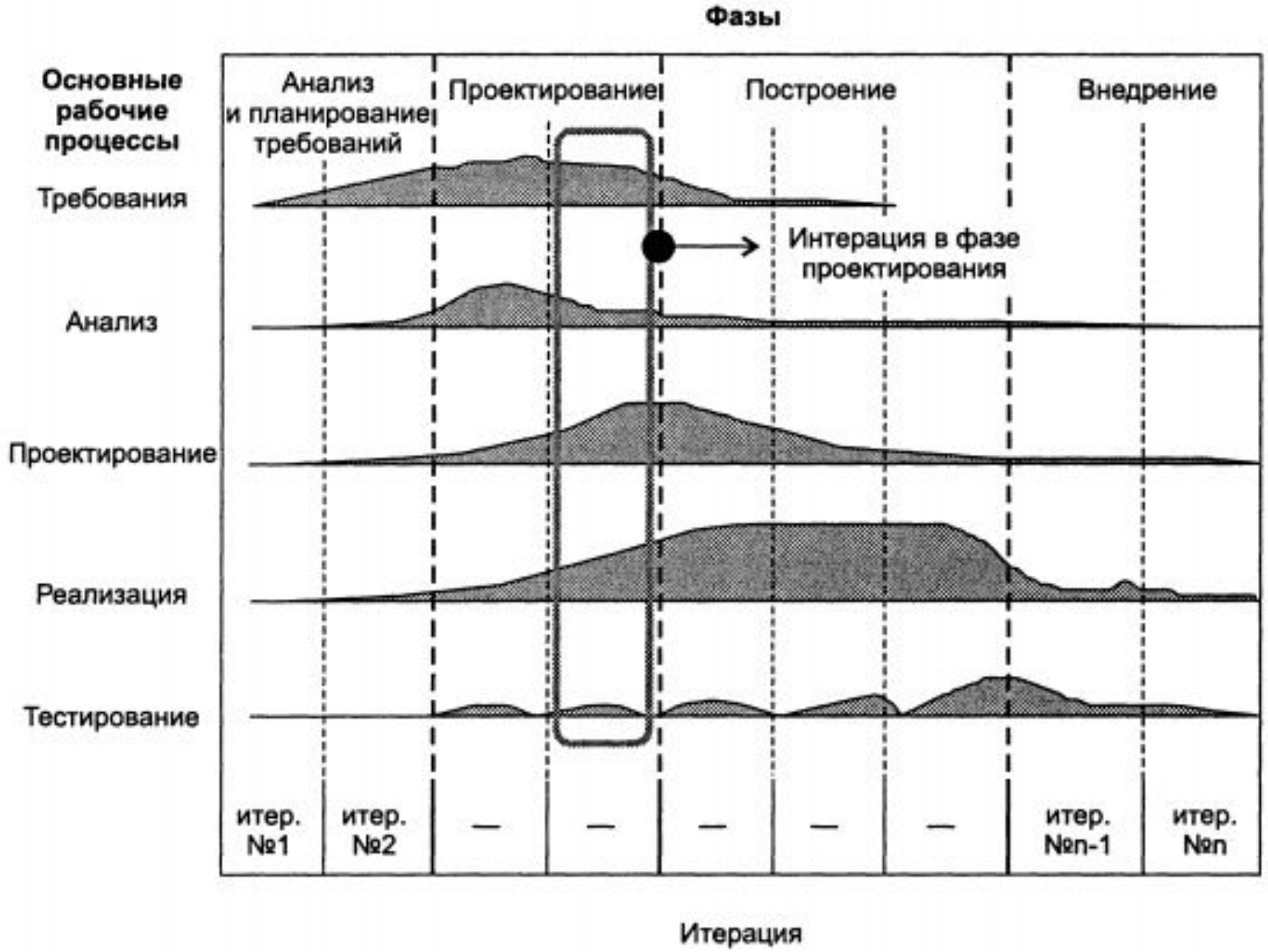


Выпуска продукта включает все артефакты



Разделение цикла разработки на фазы

- Каждый цикл осуществляется в течение некоторого времени. Это время, в свою очередь, делится на четыре фазы. В ходе смены последовательности моделей заинтересованные лица визуализируют происходящее на этих фазах. Внутри каждой фазы руководители или разработчики могут потерпеть неудачу в работе — но только на данной итерации и в связанном с ней приращении. Каждая фаза заканчивается вехой. **Веха** определяется по наличию определенного набора артефактов, например, некая модель документа должна быть приведена в предписанное состояние.
- Вехи дают руководителям возможность принять некоторые критические решения перед тем, как работа перейдет на следующую фазу.



Пять рабочих процессов — требования, анализ, проектирование, разработка и тестирование — происходят в течение четырех фаз — анализа и планирования требований, проектирования, разработки и внедрения

В ходе **фазы анализа** и планирования требований хорошая идея превращается в концепцию готового продукта, и создается бизнес-план разработки этого продукта. В частности, на этой фазе должны быть получены ответы на вопросы:

- Что система должна делать в первую очередь для ее основных пользователей?
- Как должна выглядеть архитектура системы?
- Каков план и во что обойдется разработка продукта?

На этом этапе создается пробный вариант архитектуры. Обычно он представляет собой набросок, содержащий наиболее важные подсистемы. На этой фазе выявляются и расставляются по приоритетности наиболее важные риски, детально планируется фаза проектирования и грубо оценивается весь проект.

В ходе фазы **проектирования** детально описываются большинство вариантов использования и разрабатывается архитектура системы. Архитектура определяется в виде представлений всех моделей системы, которые в сумме представляют систему целиком.

В ходе этой фазы определяются наиболее критичные варианты использования. Результатом выполнения этой фазы является **базовый уровень архитектуры.**

В конце фазы проектирования менеджер проекта занимается планированием действий и подсчетом ресурсов, необходимым для завершения проекта. Ключевым вопросом в этот момент будет следующий: достаточно ли проработаны варианты использования, архитектура и план, и взяты ли риски под контроль настолько, чтобы можно было давать контрактные обязательства выполнить всю работу по разработке?

- В ходе фазы **построения** происходит создание продукта. На этой фазе базовый уровень архитектуры разрастается до полной развитой системы. Концепции развиваются до продукта, готового к передаче пользователям. В ходе фазы разработки объем требуемых ресурсов вырастает. Архитектура системы стабильна, однако, поскольку разработчики могут найти лучшие способы структурирования системы, от них могут исходить предложения о внесении в архитектуру системы небольших изменений. В конце этой фазы продукт включает в себя все варианты использования, которые руководство и заказчик договорились включить в текущий выпуск.
- Большинство дефектов будут обнаружены и исправлены в ходе фазы внедрения.
- Ключевой вопрос окончания фазы: удовлетворяет ли продукт требованиям пользователей настолько, что некоторым заказчикам можно делать предварительную поставку?

- Фаза **внедрения** охватывает период, в ходе которого продукт существует в виде бета-выпуска или бета-версии. Небольшое число квалифицированных пользователей, работая с бета-выпуском продукта, сообщает об обнаруженных дефектах и недостатках. После этого разработчики исправляют обнаруженные ошибки и вносят некоторые из предложенных улучшений в главный выпуск, подготавливаемый для широкого распространения. Фаза внедрения включает в себя такие действия, как производство тиража, тренинг сотрудников заказчика, организацию поддержки по горячей линии и исправление дефектов, обнаруженных после поставки. Команда поддержки часто разделяет найденные дефекты на две категории: дефекты, оказывающие настолько значительный эффект на работу с программой, что это оправдывает немедленный выпуск дельта-версии, и дефекты, исправление которых можно отложить до следующего выпуска.

Рабочий процесс определения требований

- Бизнес-модели или модели предметной области для определения контекста системы.
- Модели вариантов использования для определения функциональных требований и нефункциональных требований, относящихся к отдельным вариантам использования. Модель вариантов использования определяется общим описанием, набором диаграмм и детальным описанием каждого из вариантов использования.
- Набора эскизов и прототипов интерфейса пользователя для каждого актанта, создаваемых разработчиком пользовательских интерфейсов.
- Описания обобщенных дополнительных требований, которые для конкретных вариантов использования не определяются.

Рабочий процесс анализа

- Классы анализа, их ответственности, атрибуты, связи и специальные требования. Каждый из управляющих, граничных классов или классов сущности локализует изменения (стереотипного) поведения и информацию, которую он предоставляет. Изменения пользовательских или коммуникационных интерфейсов обычно локализуются в одном или более граничных классах, а изменения долгоживущей, обычно хранимой информации, использующейся в системе, обычно содержатся в классах сущностей. Изменения же в управлении, координации, последовательности, транзакциях, а иногда и сложной бизнес-логике, затрагивающие несколько объектов (граничных и/или сущности), обычно содержатся в одном или более классах управления.
- Реализации анализа вариантов использования, описывающие, как уточнены варианты использования в понятиях кооперации в аналитической модели и ее специальных требований. Изменения реализации вариантов использования содержатся в вариантах использования, поэтому если вариант использования изменяется, то его реализация изменяется вместе с ним.
- Архитектурное представление модели анализа, включающее ее архитектурно значащие элементы. Представление архитектуры локализует изменения в архитектуре системы

Рабочий процесс проектирования

- Классы проектирования, включая активные классы и их операции, атрибуты, отношения и требования к реализации. Некоторые архитектурно значимые классы проектирования описываются на основе архитектурно значимых классов анализа. Некоторые активные классы описываются па основании классов анализа.
- Проекты реализации вариантов использования, описывающие проектирование вариантов использования в терминах кооперации внутри модели проектирования. В основном при описании проектов реализации вариантов использования в качестве спецификации используются анализы реализации вариантов использования.
- Архитектурное представление модели проектирования, включая архитектурно значимые элементы.

Рабочий процесс реализации

- Подсистемы реализации, а также их зависимости, интерфейсы и содержимое.
- Компоненты, включая файлы и исполняемые компоненты, и их взаимные зависимости. Компоненты подвергаются тестированию частей.
- Архитектурное представление модели реализации, включая архитектурно значимые элементы этой модели.

Тестирование

- Тестовые примеры, определяется предмет тестирования.
- Процедуры тестирования, определяющие условия осуществления тестовых примеров.
- Тестовые компоненты, автоматизирующие процедуры тестирования.

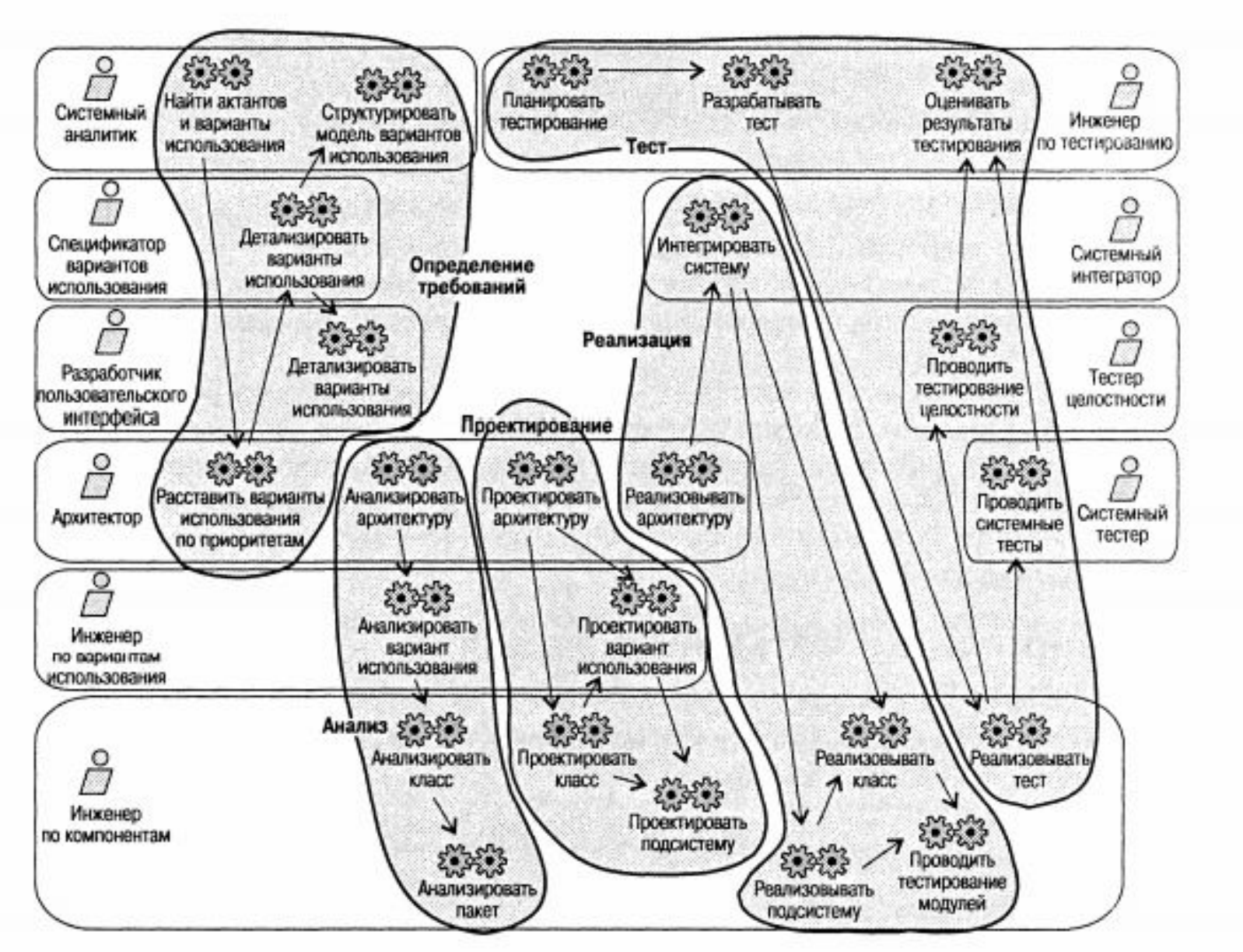
Четыре «П» в разработке программного обеспечения

- Персонал: архитекторы, разработчики, тестеры и их руководство, а также пользователи, заказчики и другие заинтересованные лица — являются исходной движущей силой программного проекта.
- Проект: организационная сущность, при помощи которой происходит управление разработкой программного обеспечения. Результатом проекта является выпущенный продукт.
- Продукт: артефакты, создаваемые в течение жизни проекта, такие как модели, тексты программ, исполняемые файлы и документация.
- Процесс: процесс создания программного обеспечения — это определение полного набора видов деятельности, необходимых для преобразования требований пользователя в продукт. Процесс служит шаблоном для создания проекта.
- Утилиты: программы, используемые для автоматизации определенных в процессе видов деятельности.

Утилиты RUP

- Rational Rose — CASE-средство визуального моделирования информационных систем, имеющее возможности генерирования элементов кода. Специальная редакция продукта — Rational Rose RealTime — позволяет на выходе получить исполняемый модуль;
- • Rational Requisite Pro — средство управления требованиями, позволяющее создавать, структурировать, устанавливать приоритеты, отслеживать, контролировать изменения требований, возникающие на любом этапе разработки компонентов приложения;
- • Rational ClearQuest — продукт для управления изменениями и отслеживания дефектов в проекте (bug tracking), тесно интегрирующийся со средствами тестирования и управления требованиями и представляющий собой единую среду для связывания всех ошибок и документов между собой;
- • Rational SoDA — продукт для автоматического генерирования проектной документации, позволяющий установить корпоративный стандарт на внутрифирменные документы. Возможно также приведение документации к уже существующим стандартам (ISO, CMM);
- • Rational Purify, Rational Quantify Rational PureCoverage, — средства тестирования и отладки:
- - Rational Purify — весьма мощное средство поиска ошибок на run-time для разработчиков приложений и компонентов, программирующих на C/C++,
- - Rational Visual Quantify — средство измерения характеристик для разработчиков приложений и компонентов, программирующих на C/C++, Visual Basic и Java; помогает определять и устранять узкие места в производительности ПО,
- - Rational Visual PureCoverage — автоматически определяет области кода, которые не подвергаются тестированию;
- • Rational ClearCase — продукт для управления конфигурацией программ (Rational’s Software Configuration Management, SCM), позволяющий производить версионный контроль всех документов проекта. С его помощью можно поддерживать несколько версий проектов одновременно, быстро переключаясь между ними. Rational Requisite Pro поддерживает обновления и отслеживает изменения в требованиях для группы разработчиков;
- • SQA TeamTest — средство автоматизации тестирования;
- • Rational TestManager — система управления тестированием, которая объединяет все связанные с тестированием инструментальные средства, артефакты, сценарии и данные;
- • Rational Robot — инструмент для создания, модификации и автоматического запуска тестов;
- • SiteLoad, SiteCheck — средства тестирования Web-сайтов на производительность и наличие неработающих ссылок;
- • Rational PerformanceStudio — измерение и предсказание характеристик производительности систем.

Роли сотрудников



- **Преимущества RUP**
- Методология **RUP** позволяет справляться с изменениями в требованиях, независимо от того, исходят они от клиента или возникают в ходе работы над проектом;
- **RUP** подчёркивает необходимость точной документации.
- Интеграция требований происходит на протяжении всего процесса разработки, и в частности в фазе построения.
- **Недостатки RUP**
- **RUP** опирается на способность экспертов и профессионалов назначить действия определённым работникам, которые затем обязаны выдать запланированные результаты в виде артефактов.
- Довольно сложный метод, который трудно внедрить в свой проект, особенно если у вас небольшая