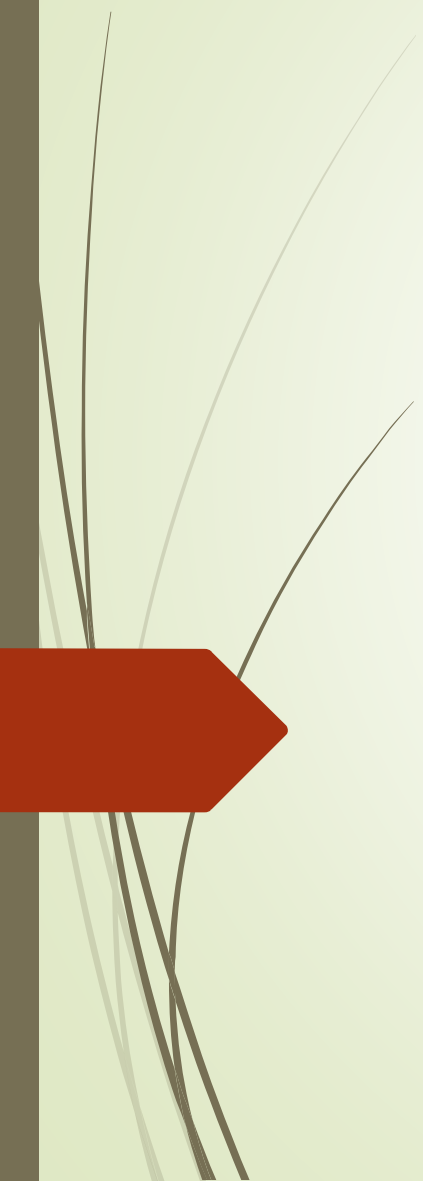




Rational Unified Process

RUP - методология разработки программного обеспечения, созданная компанией Rational Software

Общие сведения

Главная **цель** процесса проектирования и разработки программных систем состоит в создании программного продукта, обладающего высоким **качеством**, в приемлемые **сроки** в рамках прогнозируемого **бюджета**.

RUP – это процесс, направленный на поддержку **коллективной** разработки программных систем.

Все участники проекта используют единую **базу знаний**, единый **процесс**, единый **взгляд** на разработку, единый **язык моделирования**.

Все модели в RUP представляются в нотации **UML**.

RUP – это технологический **процесс** по созданию программных систем, позволяющий **улучшить** производительность коллективной разработки путём предоставления для всех этапов жизненного цикла **методик** выполнения основных видов деятельности, **шаблонов** документов, **инструкций** по работе с инструментальными средствами.



Общие сведения. Отличительные черты

- RUP – это итеративный процесс (Controlled Interactive);
- Предполагает сквозное применение аппарата Use Cases (Use Case Driven);
- Особое внимание уделяется разработке архитектуры (Architecture Centric);
- Включает управление требованиями и изменениями (Requirements Configuration and Change Management);
- Опирается на КБ концепцию разработки программных систем (Component Based Development).
- Базируется на визуальном моделировании (Visual Modeling Techniques);

Основные принципы

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

Итерации

RUP предлагает **итеративный** подход к проектированию и разработке программных систем, основанный на **спиральном** жизненном цикле.

Весь жизненный цикл включает четыре **фазы** – **вхождение** в проект (исследование), **развитие** (уточнение плана), **конструирование** и **развёртывание**.

Каждая фаза складывается из последовательности **итераций**, число которых может быть любым.

В каждой итерации **технологические процессы** последовательно применяются к разработке небольшой части программной системы.

При этом допустимо предъявление результата **заказчику**. Он имеет возможность **оценить** выполненную реализацию, выдать свои **замечания**, которые могут привести к изменению и уточнению **требований** к программной системе.

Следующая итерация предполагает **расширение** уже разработанной части путём реализации и интеграции очередной порции требований и учёта изменения требований в соответствии с замечаниями заказчика.

Преимущества итеративного подхода

- **Изменения** и уточнения требований выявляются уже в **ранний** период разработки, когда объем программного кода сравнительно небольшой, поэтому **трудоёмкость** внесения изменений существенно **ниже**.
- Уже на ранней стадии процесса разработки имеется возможность привлечения специалистов **заказчика** для **оценки** промежуточных версий (прототипов) программной системы.

Как результат – значительно более высокая вероятность того, что конечный продукт будет делать именно то, чего ждёт от него заказчик, то есть, гарантируется высокое **качество** программной системы.

Преимущества итеративного подхода

- **Снижаются** архитектурные и интеграционные **риски**.

При итеративном подходе создаётся **устойчивая** архитектура, которая прорабатывается на стадиях исследования, а затем проверяется и уточняется в нескольких начальных итерациях.

Каждая итерация предполагает интеграцию новых элементов в систему, то есть, количество интегрируемых элементов в каждой итерации невелико и легче прослеживается, чем при глобальной интеграции всех разработанных элементов программной системы.

- Итеративный подход способствует более полному **повторному использованию** программных элементов.

Анализ результатов каждой итерации позволяет архитекторам программной системы выделить фрагменты, потенциально подлежащие повторному использованию, а в следующей итерации оформить их как повторно используемые коды.

Процесс, направляемый вариантами использования

Понятие use case, введённое в UML, является **ОСНОВОЙ** для выполнения всех этапов жизненного цикла, рассматриваемых в RUP.

Понятие «business use case» (вид деятельности) является ключевым при бизнес-анализе.

На этапах анализа требований, проектирования и реализации use cases являются той «печкой», от которой «танцуют» аналитики и разработчики при выполнении проектирования и реализации программных систем.

При анализе требований, выделив ВИ, мы тем самым определяем **требование** или уровень иерархии требований к программной системе.

При детализации ВИ определяются **объекты**, и способы их **взаимодействия**, которые должны быть реализованы в программном коде.



Процесс, направляемый вариантами использования

Следует особо выделить роль ВИ в **планировании** итераций.

Как определить, какая функциональность должна быть реализована в очередной итерации?

Здесь на выручку приходят **диаграммы** ВИ.

Каждому ВИ можно приписать **приоритет**, определяющий, в какой итерации его следует реализовать.

Можно показать все ВИ, реализуемые в очередной **итерации**, на отдельной диаграмме (или нескольких диаграммах).



Процесс, основанный на архитектуре

Разработка архитектуры программной системы в RUP преследует следующие цели:

- **Описание** организации программной системы;
- **Определение** компонентов и их интерфейсов;
- **Определение** подсистем и объединение компонентов в подсистемы;
- **Создание** основы для повторного использования компонентов;
- **Создание** базиса для разработки;
- **Контроль** над проектом и поддержка целостности системы.

Процесс, основанный на архитектуре

Архитектура программной системы находит отражение в различных **архитектурных представлениях**.

- **Логическое представление** отражает функциональные требования к системе. Оно определяет основные (архитектурно значимые) пакеты, подсистемы и классы проекта.
- **Реализационное представление** описывает организацию статических программных модулей (компонентов, файлов данных, исходного кода и др.) в терминах пакетов и уровней.
- **Процедурное представление** отражает аспекты параллельности задач, потоков и процессов во время работы системы и их взаимодействия.
- **Представление развёртывания** показывает, как компоненты отображаются на базовые платформы и вычислительные узлы.
- **Use case представление** содержит ключевые ВИ и сценарии.

Процесс, основанный на архитектуре

Архитектурные представления акцентируют внимание только на таких элементах, которые имеют **существенное влияние** на структуру программной системы, её производительность, масштабируемость, устойчивость, сопровождаемость, возможность развития.

К числу таких элементов относятся:

- Классы, моделирующие основные объекты деятельности;
- Механизмы, определяющие поведение этих классов;
- Уровни и подсистемы;
- Интерфейсы;
- Основные процессы и управляющие потоки.
- В RUP предусмотрено создание отдельного документа, содержащего описание архитектуры программной системы.

Технологические процессы RUP

Определено 9 **технологических процессов**, для каждого из которых предложена методика выполнения. Технологические процессы делятся на две категории – основные процессы и процессы поддержки. К основным относятся:

- Бизнес-анализ.
- Управление требованиями.
- Анализ и проектирование.
- Реализация.
- Тестирование.
- Развёртывание.

Технологические процессы RUP

Вспомогательные процессы включают:

- Управление проектом,
- Управление конфигурацией,
- Управление средой.

Для каждого технологического процесса предусмотрены:

роли, определяющие поведение и обязанности отдельных лиц и групп, работающих в одной команде (например, системный аналитик, тестировщик),

виды деятельности, определяющие работы, выполняемые исполнителями (например, проектирование класса, проектирование ВИ) и

артефакты – документы, используемые, порождаемые или модифицируемые процессом. Основные артефакты в RUP – модель, элемент модели, документ, исходный код, исполняемая программа.

Жизненный цикл

Полный жизненный цикл разработки продукта состоит из четырех фаз, каждая из которых включает в себя одну или несколько итераций:

1. Начало (Inception)

На этом этапе:

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

При завершении фазы «Начало» оценивается достижение **вехи** целей жизненного цикла «Lifecycle Objective Milestone», которое предполагает **соглашение** заинтересованных сторон о продолжении проекта.

Жизненный цикл

2. Проектирование (Elaboration)

На этапе «Проектирование» производится **анализ** предметной области и построение исполняемой **архитектуры**. Это включает в себя:

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

Успешное выполнение фазы «Проектирование» означает достижение **вехи** архитектуры жизненного цикла (Lifecycle Architecture Milestone).

Жизненный цикл

3. Построение (Construction)

Во время этой фазы происходит **реализация** большей части функциональности продукта. Фаза «Построение» завершается первым внешним **релизом** системы и **вехой** начальной функциональной готовности (Initial Operational Capability).

4. Внедрение (Transition)

Во время фазы «Внедрение» создаётся **финальная** версия продукта и передаётся от разработчика к заказчику.

Это включает в себя программу **бета-тестирования**, **обучение** пользователей, а также определение **качества** продукта.

В случае, если качество **не соответствует** ожиданиям пользователей или критериям, установленным в фазе «Начало», фаза «Внедрение» **повторяется** снова. **Выполнение** всех целей означает достижение **вехи** готового продукта (Product Release) и **завершение** полного цикла разработки.

Обследование организации (бизнес-анализ)

Цели

- **понять** структуру и динамику работы организации;
- **определить** проблемы, возникающие в работе организации, и возможности их решения, направленного на повышение эффективности работы;
- **гарантировать**, что заказчики, конечные пользователи и разработчики будут иметь одинаковое понимание деятельности организации;
- **вывести** требования к программным системам, автоматизирующим работу организации.

Организация описывается как с **внешней** точки зрения – какие результаты предоставляются её клиентам, так и с **внутренней** – роли, и их связи с деятельностью организации.

Эта информация служит системным аналитикам в качестве **связующей** при определении **требований** к программной системе.



Обследование организации (бизнес-анализ)

Бизнес-анализ вовсе **не** является **обязательным** для каждого проекта разработки программной системы.

Если заказчик имеет хорошо **отлаженный** производственный **цикл**, использует программные средства **автоматизации**, точно **представляет** себе, какие производственные задачи должна решать новая программная система в дополнение к уже автоматизированным, то проведение бизнес-анализа может не потребоваться.

Основным результатом бизнес-анализа является **бизнес-модель**, которая представляется на UML.

Обследование организации (бизнес-анализ)

Роли

В моделировании бизнеса участвуют:

- **бизнес-аналитик** – специалист организации-разработчика, который возглавляет и координирует работы по моделированию бизнеса;
- **бизнес-разработчик** – специалист организации-разработчика, который детализирует и уточняет бизнес модели, определяет бизнес-исполнителей их обязанности и действия;
- **заинтересованные лица** – люди, предоставляющие информацию. Это могут быть бизнес-исполнители или клиенты организации, а также прочие люди, заинтересованные как в собственно результатах моделирования, так и в будущей программной системы.
- **эксперт – представитель** обследуемой организации, участвующий в разработке модели (консультации, организация встреч с заинтересованными лицами, оценка результатов). Эксперт, в частности, может быть одним из бизнес-исполнителей.

Обследование организации (бизнес-анализ)

Артефакты

При моделировании создаются следующие артефакты в виде текстовых документов и моделей, описанных на UML:

- Документ «**Видение бизнеса**» – определяет цели проведение бизнес-анализа.
- **Структура организации** – статическое описание подразделений организации и отношений подчинённости в виде диаграмм пакетов и/или классов.
- Модель **видов деятельности** включает бизнес-актёров и виды деятельности организации.

К числу бизнес-актёров относятся: заказчики, партнёры, поставщики, власти (представители закона, инспекция и др.), дочерние организации, собственники и инвесторы, внешние информационные системы.

Бизнес-актёры помогают определить границы организации, которую требуется описать.

Виды деятельности представляют собой бизнес-процессы.

Модель видов деятельности представляется с помощью use case диаграмм.

Обследование организации (бизнес-анализ)

Артефакты

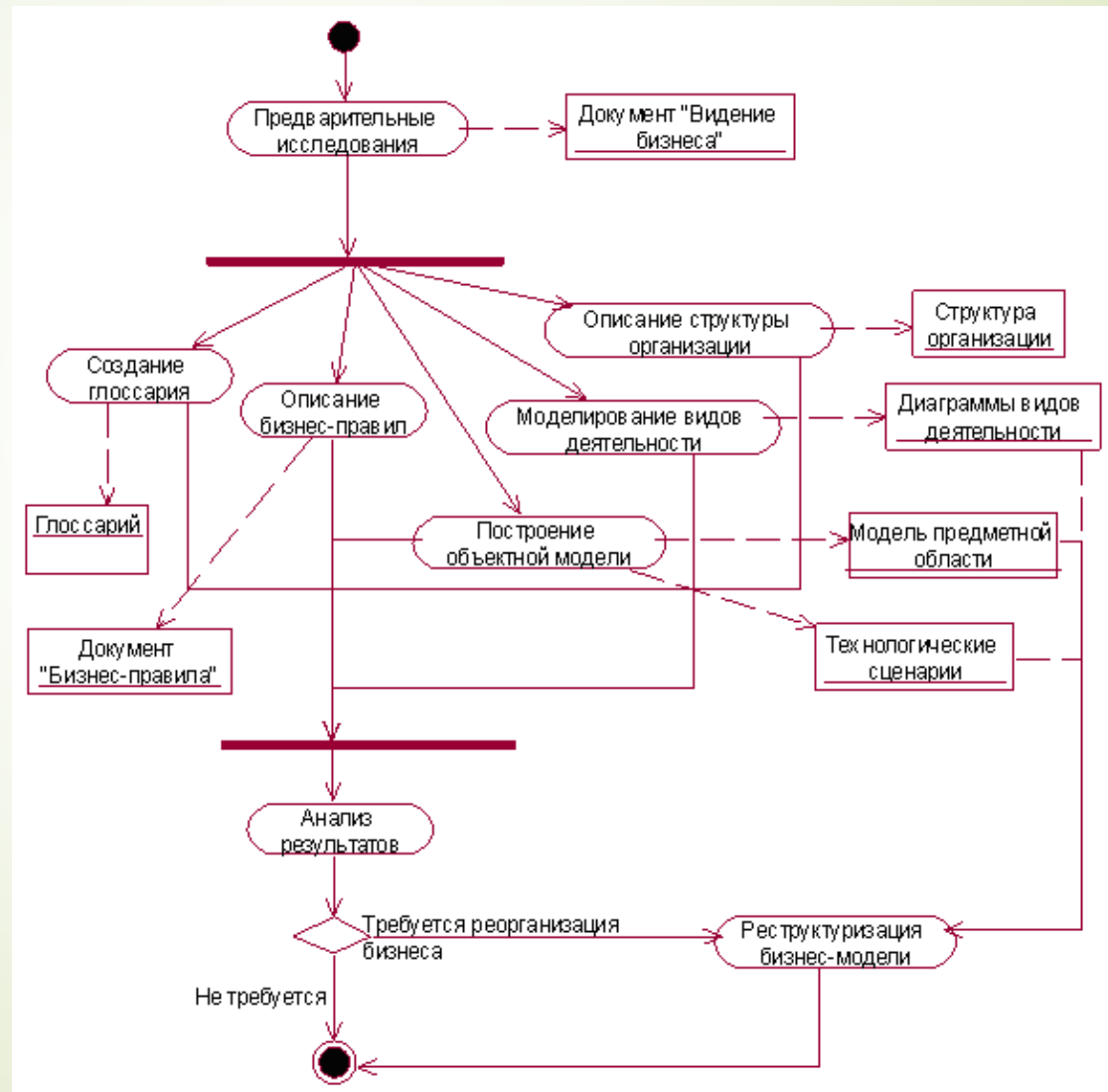
- **Объектная** модель включает бизнес-актёров, бизнес-исполнителей и бизнес-сущности, а также содержит описание их взаимодействий при реализации видов деятельности.

Модель представляется на UML с помощью диаграмм классов и взаимодействий (последовательностей, коопераций, деятельностей), которые иногда называют технологическими сценариями.

- Модель **предметной области** является подмножеством объектной модели. Она описывает основные бизнес-сущности и связи между ними. Эта модель представляется в виде диаграмм классов.
- **Глоссарий** – текстовый документ, содержащий определения основных понятий, используемых в данном бизнесе.
- **Оценка** деятельности организации – текстовый документ, описывающий текущее состояние организации, в которой будет использоваться программной системы.
- **Бизнес-правила** – текстовый документ, определяющий условия и ограничения, которым должен удовлетворять бизнес.
- Дополнительные **спецификации** – текстовый документ, содержащий описание свойств бизнеса, не включённых в бизнес-модель.

Обследование организации (бизнес-анализ)

Процесс бизнес-анализа





Обследование организации (бизнес-анализ)

Процесс бизнес-анализа

Построение всех предписываемых проекций модели бизнеса выполняется **параллельно**.

Не всегда **требуется** создавать все проекции. В частности, иногда достаточно просто построить модель предметной области.

Решение о составе модели принимает бизнес-аналитик.

При построении бизнес-модели используют нормативные **документы** организации (устав, штатное расписание и др.), а также **информацию**, предоставляемую заинтересованными лицами, для чего проводятся интервью и совещания, заполняются анкеты и опросные листы.

Обследование организации (бизнес-анализ)

Процесс бизнес-анализа

- Созданная в итоге бизнес-модель является **основой** для последующего моделирования программной системы.
- Например, модель **видов деятельности** преобразовывается в модель **ВИ**.
- Такое преобразование может быть **формализовано**. Необходимо выделить те виды деятельности, которые подлежат **автоматизации**, и объявить их вариантами использования программной системы, а также преобразовать бизнес-исполнителей в актёров, поскольку они являются внутренними сущностями организации, но внешними по отношению к системе.
- Модель предметной области включается как составная часть в логическую модель программной системы, а технологические сценарии являются источником для определения потоков событий в ВИ.

Реинжиниринг программных систем

- Один источник требований к системе – это **бизнес-анализ**, проводимый в начальной фазе проекта.
- Второй источник – **программные системы**, эксплуатируемые в организации. Разработка современных программных систем редко когда начинается "с нуля".
- Чаще всего у заказчика имеется работающая (**наследуемая**) система, для которой требуется **расширить** состав выполняемых функций, перевести её на другую, более современную платформу, повысить производительность и т. д.
- При этом типичной является ситуация, когда наследуемая система **плохо документирована** (нет моделей и описания, частично или полностью отсутствуют исходные коды программ, устарела пользовательская документация и др.).
- В этом случае прежде чем браться за разработку новой системы, необходимо **документировать наследуемую**.

Реинжиниринг программных систем

- В терминах RUP это означает, что необходимо построение **модели**, описывающей программную систему с различных точек зрения.
- Моделирование наследуемой системы основывается на проведении **реинжиниринга**, в процессе которого выполняется восстановление описывающих систему моделей на основе анализа имеющейся информации.
- При этом функциональность, реализованная в наследуемой системе, определяет часть **требований** к проектируемой программной системе (в большинстве случаев достаточно значительную).



Реинжиниринг программных систем

Цели реинжиниринга

- получение представления о составе существующей системы;
- моделирование существующей системы;
- определение фрагментов программного кода, которые могут быть использованы в разрабатываемой новой системе;
- определение наследуемых данных.



Реинжиниринг программных систем

Задачи реинжиниринга

- определение системных архитектур;
- определение актёров существующей системы;
- определение функциональности существующей системы;
- определение логической структуры системы;
- восстановление реляционной модели данных.

Реинжиниринг программных систем

Порядок решения этих задач принципиально **отличается** от порядка действий, выполняемых при разработке новых проектов.

Логическая архитектура системы определена её **реализацией**, в частности, структурой каталогов, размещением файлов по каталогам, распределением задач между сервером и клиентскими местами.

Кроме того, часть моделей системы может быть получена **автоматически** с помощью инструментальных средств.

Таким образом, не функциональное описание системы служит основой для выявления классов и отношений между ними, а наоборот, предварительно полученные диаграммы **классов** могут существенно **облегчить** описание поведения системы.

Реинжиниринг программных систем

Начальная фаза

- Начать процесс реинжиниринга следует с **определения** того, что есть по существующей системе (исходные тесты, БД, описания и т. д.).
- При этом **фиксируется** текущее состояние наследуемой системы (все изменения, вносимые в неё после этого момента, при выполнении реинжиниринга не учитываются).
- Если есть возможность выполняется **развёртывание** наследуемой программной системы у разработчика.

Реинжиниринг программных систем

Определение системных архитектур

- Работы по описанию архитектур начинаются фактически на начальном этапе, когда определяется состав оборудования и стандартное программное обеспечение (ПО), необходимые для инсталляции и запуска существующей зафиксированной системы.
- Тем самым фактически определяются архитектуры БД, оборудования и стандартного ПО, телекоммуникаций.
- Все архитектуры представляются в нотации UML и при необходимости дополняются текстовыми описаниями.
- Построенные архитектурные модели в процессе реинжиниринга будут уточняться и дополняться.

Реинжиниринг программных систем

Автоматический реинжиниринг

- Автоматический реинжиниринг осуществляется с помощью **инструментальных** средств визуального **моделирования**. Его выполнение позволяет построить модели, которые могут быть приняты как исходные. Автоматическому реинжинирингу подвергается как бизнес логика (если есть исходные коды на объектно-ориентированном или объектно-базированном языке), так и БД.
- Автоматический реинжиниринг бизнес логики может быть выполнен только в случае, когда имеются (полностью или частично) **исходные тексты** программ. В результате автоматического реинжиниринга кодов создаются диаграммы классов и диаграммы компонент UML.
- Реинжиниринг БД выполняется с помощью **инструментальных** средств **проектирования** БД. Результатом является реляционная модель данных, которая может графически отображаться этим средством. Полученная реляционная модель может по усмотрению разработчиков переведена в диаграмму классов UML путём использования применяемого инструментального средства разработки БД или программных мостов со средствами визуального объектно-ориентированного моделирования.

Реинжиниринг программных систем

Редактирование диаграмм моделей

- Модели, полученные **автоматически**, весьма **сложно читать** и **анализировать**, поскольку элементы модели размещаются без учёта наглядности диаграмм. Поэтому построенные модели должны быть **отредактированы**. В процессе редактирования не следует выполнять содержательные преобразования (удалять или добавлять элементы модели). Главной целью **редактирования** на этом этапе является достижение **наглядности** диаграмм. Для этого используется перемещение элементов диаграмм. В процессе редактирования могут вноситься комментарии к элементам модели. Например, можно прокомментировать назначение отдельных классов. Комментарии заносятся в поля спецификаций элементов моделей.
- Если диаграмма содержит слишком много элементов, то анализировать её сложно.

Попробуйте проанализировать диаграмму, содержащую более **100** классов! Поэтому целесообразно разбить такую диаграмму на несколько отдельных диаграмм, оставляя в каждой примерно по **7 – 10** элементов.

Реинжиниринг программных систем

Метод повышения **наглядности** диаграмм хорошо известен. Это иерархическая **реструктуризация**. Средством её осуществления в UML являются **пакеты**.

Сложные программные системы обычно включают несколько **подсистем**, имеющих разное целевое назначение и функциональность. Поэтому на верхнем уровне иерархии можно показать **пакеты – подсистемы**.

Каждый из таких пакетов должен получить **имя**, отражающее **суть** соответствующей подсистемы. На этом уровне можно также показать пакеты классов, являющиеся общими для всей системы и используемые подсистемами.

На начальной стадии реструктуризации логической модели можно ввести пакет **верхнего уровня**, куда помещаются классы, которые трудно отнести к какому-либо другому пакету. В любой программной системе есть пользовательский интерфейс, связь с БД, управление, обработка, классы данных. Такого типа пакеты можно ввести в каждой подсистеме на следующем уровне иерархии.

Реинжиниринг программных систем

Построение функциональной модели

Модель **функционирования** описывается с помощью диаграмм **ВИ** и детализирующих их диаграмм **последовательностей** и **деятельностей**.

Источником для её построения является работающая **наследуемая** система и проводимые с ней эксперименты. На этом этапе особенно эффективно привлечение к работам по реинжинирингу **эксперта** организации-заказчика.

С его помощью можно быстрее и точнее определить состав актёров наследуемой системы и основные ВИ. Эксперт заказчика может словесно **описать**, кто использует систему и что она должна делать для пользователей каждого типа. Он может также **информировать**, с какими другими системами взаимодействует наследуемая программная система, какие работы осуществляются периодически.

Все эти сведения будут способствовать более **точному** пониманию функциональности системы разработчиками

Реинжиниринг программных систем

Определение актёров

- Кто является непосредственными пользователями системы? Необходимо при ответе на данный вопрос указывать роли, а не конкретных людей, исполняющих эти роли.
- С каким внешним оборудованием или программами осуществляет взаимодействие система?
- Выполняет ли система работы, активизируемые наступлением конкретного времени или истечением определённых интервалов времени (при положительном ответе одним из актёров является таймер)?
- Ответы на поставленные вопросы можно получить либо путём опроса экспертов заказчика, либо из документации на систему, либо (если таковых не имеется) путём запуска системы и анализа экранных форм или меню.
- Актёрам следует присвоить имена, отражающие их роли в работе с системой.

Реинжиниринг программных систем

Определение вариантов использования

- Определение **ВИ** выполняется на основе анализа **экранов** работающей системы. Сначала определяются пакеты ВИ. Для этого следует найти все первичные экраны и для каждого из них в модели на головной диаграмме ВИ завести отдельный пакет. Таким образом, на этой диаграмме будет пакет актёров и несколько пакетов ВИ.
- Далее выполняется **детализация** построенных пакетов ВИ на основе анализа экранных форм. Рекомендуемые правила:
 - если экран содержит меню, то это пакет ВИ. При этом каждая строка меню – это либо подпакет, либо отдельный ВИ;
 - если основной экран – форма, то это отдельный ВИ.



Реинжиниринг программных систем

Определение взаимодействий актёров и ВИ

- Поскольку очень важно показать, как актёры соотносятся с ВИ, после нахождения ВИ определяется, какие актёры взаимодействуют с системой в этом варианте.
- В модель включаются ассоциации.
- Они имеют направления, соответствующие направлениям передачи информации между актёрами и ВИ.

Реинжиниринг программных систем

Распределение по пакетам

Если число актёров или ВИ слишком **велико**, то для упрощения поддержки модели ВИ целесообразно разделить их по **пакетам**.

Это также **упрощает** понимание модели и распределение ответственности путём назначения разработчиков, ответственных за пакеты ВИ.

Можно использовать следующие **критерии** упаковки ВИ в один пакет:

- если они взаимодействуют с одним актёром;
- имеют друг с другом отношения включения или расширения;
- Можно использовать и другие способы обеспечения наглядности модели, важно лишь иметь чёткую стратегию разбиения на пакеты.

Реинжиниринг программных систем

Построение навигации экранов

- Одновременно с выделением ВИ строится **навигация** экранов наследуемой системы в виде диаграммы **классов** UML.
- Каждый **экран** показывается в модели как отдельный **класс**, в котором **поля** соответствуют **атрибуты**, функциональным **кнопкам** – **операции**, а кнопкам **меню** – одноименные **отношения**.

Реинжиниринг программных систем

Детализация функциональности

- Детализация функциональности представляет собой построение сценариев реализации ВИ, представленных в модели ВИ.
- Она выполняется с помощью диаграмм последовательностей и диаграмм деятельности UML.
- Выбор вида диаграмм в каждом конкретном случае зависит от того, что преобладает в данном ВИ – логика выполнения или передачи данных.
- В первом случае предпочтительно применять диаграммы деятельности, где легко показывать ветвления и параллельную обработку, во втором – диаграммы последовательностей.

Реинжиниринг программных систем

Детализация функциональности

- Детализация требуется в особенности для тех вариантов использования, в которых важна **последовательность** действий, учитывается состояние системы, имеется разветвлённая логика выполнения.
- Целесообразно детализировать выполнение ВИ, определяющих **основную функциональность** системы.
- Если заказчик высказывает пожелания относительно того, что из наследуемой системы должно быть **сохранено** в новой программной системе, то именно эти ВИ должны быть **детализированы**.

Реинжиниринг программных систем

Детализация функциональности

- **Детализация** осуществляется на основе анализа **ИСХОДНЫХ** кодов. По текстам программ выявляются ветвления, выражения, циклы. Это позволяет восстановить алгоритмы, представив их в виде диаграмм деятельности или диаграмм состояний. Другой путь – это проведение **экспериментов** с работающей наследуемой **системой**. Варьирование входных данных и анализ реакции системы на эти данные делает возможным обнаружение ветвлений и ограничений. Можно также выдвигать и проверять гипотезы об алгоритмах вычислений.
- **Модели**, построенные в результате **реинжиниринга**, являются **основой** для определения требований к проектируемой программной системе, а также для построения логической и функциональной моделей новой системы.

Управление требованиями

Требования к программной системе

Требование – это характеристика или условие, которому должна удовлетворять программная система.

Функциональные требования определяют действия, которые должна выполнять программная система, без учёта физических ограничений. Тем самым они определяют поведение системы при вводе и выводе. Процесс **выявления** функциональных требований весьма **сложный** и **трудоёмкий**. Это объясняется следующими причинами:

- таких требований к системе обычно **много**,
- заказчик не всегда способен **чётко сформулировать**, чего он хочет от системы,
- **требования** в итоговом документе должны быть изложены так, чтобы они **одинаково** понимались заказчиком и исполнителем и не допускали неоднозначности,
- между функциональными **требованиями** могут быть разные **зависимости**, усложняющие управление ими в случае необходимости внесения изменений.

Управление требованиями

Требования к программной системе

Для преодоления этих трудностей применяется **моделирование** требований.

Модель требований позволяет:

- **во-первых**, установить иерархию требований, что способствует лучшему пониманию человеком,
- **во-вторых**, даёт наглядное графическое представление требований и зависимостей между ними,
- **в-третьих**, позволяет связать графическую форму представления с текстовой, обеспечивая человека полной информацией.

Управление требованиями

Требования к программной системе

- **Нефункциональные требования** не описывают поведение программной системы, но описывают её атрибуты или атрибуты окружения.
- Непункциональные требования не требуется включать в модель требований, но они должны быть точно сформулированы.
- Обычно нефункциональных требований не бывает много, однако они кардинальным образом влияют на выбор архитектуры системы.

Управление требованиями

Требования к программной системе

Управление требованиями – это достаточно сложный и растянутый во времени процесс. Он продолжается в течение большей части жизненного цикла, поскольку изменения могут вноситься как во время разработки, так и после сдачи системы на этапе опытной эксплуатации и при сопровождении. Причины этого заключаются в том, что требования:

- неочевидны,
- исходят из многих источников,
- трудно формулируются (язык неоднозначен),
- состоят из множества различных деталей,
- неравнозначны,
- связаны друг с другом,
- лежат не только в функциональной области.
- могут изменяться в течение разработки и при сопровождении.

Управление требованиями

Цели анализа и моделирования требований

- **достижение** соглашения между разработчиками, заказчиками и пользователями о том, что должна делать программная система;
- **достижение** лучшего понимания разработчиками того, что должна делать система;
- **ограничение** системной функциональности;
- **создание** базиса для планирования разработки проекта;
- **определение** пользовательского интерфейса;
- **составление** спецификации требований.

Управление требованиями

Роли

- **Системный аналитик** – специалист организации-разработчика, который возглавляет и координирует работы по выявлению и моделированию требований.
- **Разработчик** ВИ – специалист организации-разработчика, который детализирует и уточняет модели требований.
- **Заинтересованные лица** – люди, предоставляющие информацию.
- **Эксперт** – представитель заказчика, участвующий в разработке модели требований.
- **Разработчик** пользовательского интерфейса – специалист организации-разработчика, который создаёт прототип пользовательского интерфейса программной системы.

Управление требованиями

Артефакты

Для достижения поставленных целей предусматривается создание следующих **документов**:

- **Предварительное соглашение** – текстовый документ, который описывает, что будет включено в программную систему и что решено исключить, то есть, он ограничивает системную функциональность.

В нём отражаются пожелания заинтересованных лиц, а также указываются основные нефункциональные требования (например, описывается платформа реализации, точность вычислений, время отклика).

Управление требованиями

Артефакты

Для достижения поставленных целей предусматривается создание следующих **документов**:

- **Модель требований** служит для достижения соглашения между заказчиком и разработчиками, давая возможность заказчику убедиться в том, что система будет делать то, что они ожидают, а разработчикам создать то, что требуется.

Модель требований позволяет, **во-первых**, установить иерархию требований, что способствует лучшему пониманию человеком, **во-вторых**, даёт наглядное графическое представление требований и зависимостей между ними, **в-третьих**, позволяет связать графическую форму представления с текстовой.

Модель включает актёров и ВИ, показывает, как система взаимодействует с актёрами и что она делает в каждом ВИ.

Управление требованиями

Артефакты

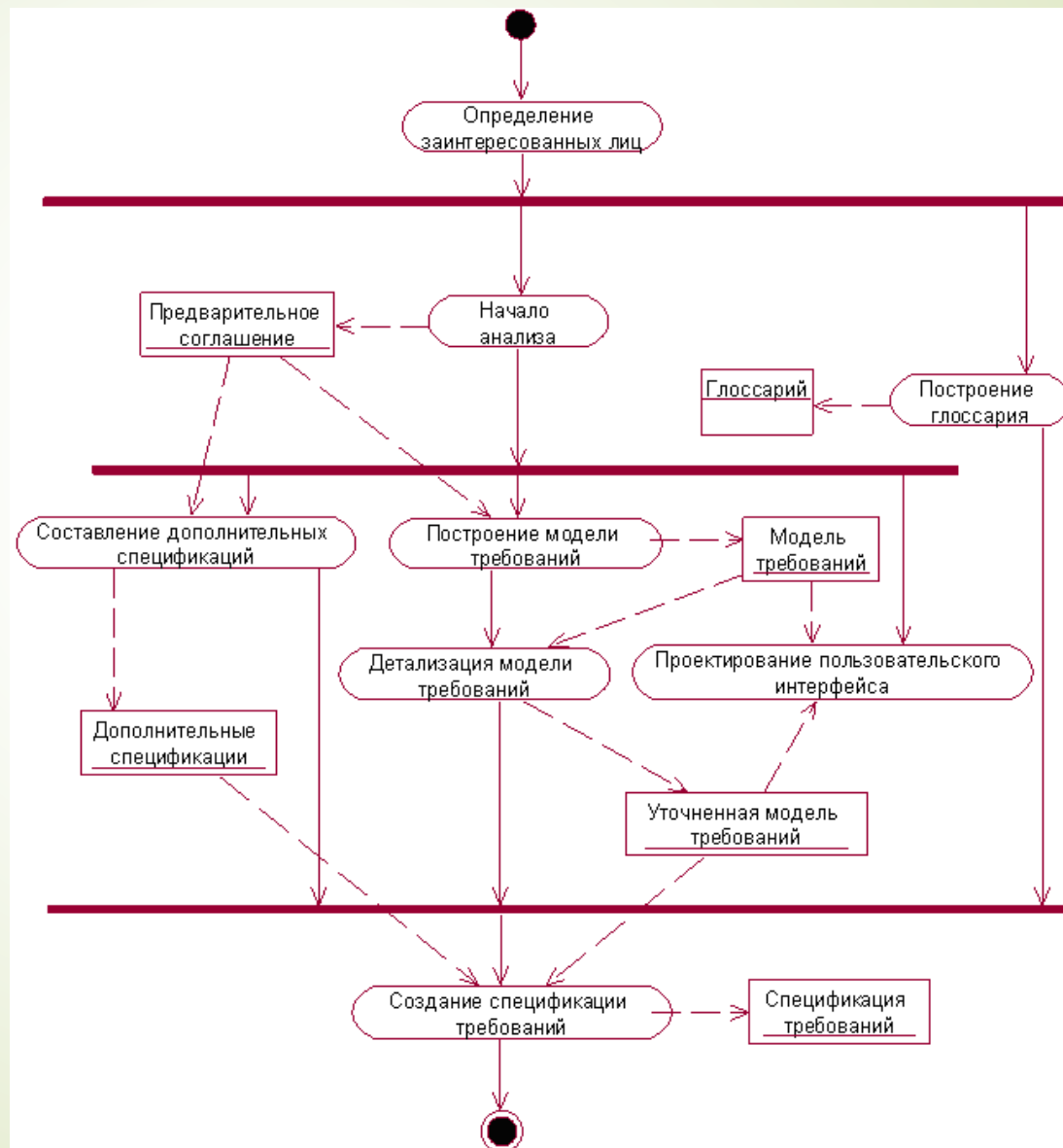
Для достижения поставленных целей предусматривается создание следующих **документов**:

- **Спецификация требований** (Software Requirements Specification) – основной документ, используемый при проектировании и разработке программной системы. Она включает модель требований и дополнительные спецификации, которые представляют собой текстовое описание требований к конечному продукту, но не к процессу его разработки и не содержат деталей реализации требований.
- **Прототип пользовательского** интерфейса обеспечивает визуальное представление интерфейса пользователя с программной системой.
- **Глоссарий** – текстовый документ, содержащий определения основных понятий и терминов, которые должны одинаково пониматься заказчиком и разработчиком. Источниками данных для создания перечисленных артефактов могут, в частности, служить артефакты, созданные при бизнес-анализе.

Управление требованиями

Процесс

В процессе анализа и моделирования требований можно выделить несколько основных этапов:



Управление требованиями

Начало анализа

- Сначала следует определить **круг** заинтересованных **лиц**.

Если проводился бизнес-анализ, то они уже известны.

Собираются пожелания заинтересованных лиц к будущей программной системе.

Эти пожелания анализируются, определяются основные свойства и границы программной системы, достигаются соглашения о том, какие проблемы должны быть решены.

Управление требованиями

Начало анализа

- Результаты. Должен быть составлен документ «Предварительное соглашение», который будет являться отправной точкой для выполнения всех последующих работ.

На этом этапе начинается создание глоссария.

Если глоссарий был создан при бизнес-анализе, то он используется как отправной документ.

Поскольку здесь речь идёт о выявлении требований к программной системе, в глоссарий могут включаться термины, относящиеся к реализации (например, браузер, сервер и др.).

Определения этих терминов должны способствовать лучшему пониманию системы заинтересованными лицами.

Управление требованиями

Построение модели требований

- Эта работа предполагает **выявление** актёров, ВИ и **взаимодействий** между ними.

Если было проведено обследование организации, то в качестве источника для определения актёров и ВИ может служить бизнес-модель.

Управление требованиями

Построение модели требований

- Если число ВИ слишком **велико**, то для упрощения читаемости и поддержки модели целесообразно **разделить** их по пакетам.

Это также упрощает понимание модели и распределение ответственности путём назначения разработчиков, ответственных за пакеты ВИ.

Пакеты позволяют организовать иерархию требований. Верхний уровень иерархии обычно определяется, исходя из основных видов деятельности организации.

Если был выполнен бизнес-анализ, то основные виды деятельности уже представлены в бизнес-модели в виде пакетов, и они могут быть просто скопированы в модель требований.

Пакетов верхнего уровня не должно быть много. Целесообразно выделить пакет администрирования и пакет вспомогательных действий, и **2 – 4** пакета, основных видов деятельности. В свою очередь, пакеты верхнего уровня могут включать пакеты следующего уровня и т. д.

Управление требованиями

Построение модели требований

- После формирования исходной модели требований, нужно внимательно проверить, что она покрывает все заявки заинтересованных лиц, представленные в документе «Предварительное соглашение».

Управление требованиями

Детализация модели требований

Цели данной деятельности:

- **извлечение** из ВИ характерных фрагментов, которые могут рассматриваться как отдельные абстрактные ВИ. Примерами таких частей могут быть общие фрагменты, необязательные фрагменты и исключения;
- **обнаружение** новых абстрактных актёров, которые играют роли, разделяемые несколькими актёрами;
- **реструктуризация** модели требований;
- детальное **описание** потоков событий для ВИ;
- **задание** приоритетов ВИ.

Управление требованиями

Детализация модели требований

Когда работа продвинулась достаточно далеко, может появиться желание пересмотреть исходную модель, поскольку существует тенденция вводить на первых порах слишком **много актёров**. Следует помнить, что любая **модификация актёров** может вызвать существенные **изменения** в системном **интерфейсе** и **поведении**.

Детализация ВИ предполагает разработку диаграмм последовательностей, коопераций или деятельности, описывающих выполнение основной и вспомогательных работ в конкретном ВИ.

Управление требованиями

Детализация модели требований

Определение приоритетов означает, что все ВИ **ранжируются** на критичные, важные и вспомогательные.

Критичные ВИ, представляют основную системную **функциональность** или имеют существенное архитектурное значение.

Важные ВИ определяют такие системные функции, как **сбор** статистики, **составление** отчётов, **контроль** и функциональное **тестирование**.

Если они не реализованы, то система может **выполнять** своё предназначение, но со значительно **худшим** качеством сервиса.

Вспомогательные ВИ определяют **удобство** использования программной системы.

Управление требованиями

Составление дополнительных спецификаций

- Дополнительные спецификации представляют собой **текстовые** описания требований.
- Они **дополняют** модель требований и наряду с ней включаются в итоговый документ – спецификацию требований к программной системе.
- Следует чётко понимать, что каждый **ВИ** есть некоторое иерархическое **требование** к программной системе.
- Очень важно, чтобы между **моделью** требований и **описанием** требований в спецификации было точное **соответствие**.
- Если ВИ простой, то в спецификации он описывается как отдельное функциональное требование.
- В более сложных случаях ВИ может быть разбит на несколько более простых вариантов (на диаграммах это можно сделать с помощью пакетов, введя ещё один уровень иерархии, в спецификации – ввести следующий уровень нумерации).

Управление требованиями

Проектирование пользовательского интерфейса

Этот процесс выполняется для того чтобы заказчик мог более **точно представить** себе работу и **возможности** будущей программной системы и выдать свои замечания и уточнения требований. В зависимости от сложности проекта и уровня подготовленности заказчика результаты этих работ могут быть представлены в различных формах:

- программная реализация, воспроизводящая точный вид экранных окон;
- альбом экранных форм;
- модель навигации экранов в виде диаграмм классов с указанием атрибутов – полей и операций – кнопок.

Управление требованиями

Создание спецификации требований

- Спецификация требований создаётся на основе модели требований и дополнительных спецификаций.
- Она утверждается руководством заказчика и разработчика и служит основным отправным документом для проектирования и разработки.
- В частности, модель требований, входящая в неё, в дальнейшем будет развита в модель анализа и дизайна.

Анализ и проектирование

Цель и задачи анализа и проектирования

- Цель процесса анализа и проектирования состоит в разработке технических инструкций, предписывающих, как реализовать программную систему, удовлетворяющую сформулированным требованиям.
- Для этого следует хорошо понять требования к программной системе и преобразовать их в проект системы, выбрав правильную стратегию реализации.
- На ранних стадиях процесса должна быть создана устойчивая архитектура, на основе которой можно спроектировать программную систему, лёгкую для понимания, построения и развёртывания.
- Архитектура должна быть согласована со средой реализации с целью удовлетворения требований к производительности, устойчивости, безопасности, расширяемости и тестируемости.

Анализ и проектирование

Цель и задачи анализа и проектирования

К числу решаемых задач при этом относятся:

- разработка точной архитектуры распределённой программной системы;
- преобразование модели требований в модель проектную разрабатываемой системы;
- адаптация проекта системы к среде реализации с целью повышения производительности разработки;
- выбор механизмов реализации и определение ограничений на реализацию;
- разработка компонентной структуры;
- распределение компонентов по узлам.

Анализ и проектирование

Цель и задачи анализа и проектирования

Главной задачей анализа является преобразование требований в форму, понятную разработчику, то есть, определение подсистем, компонентов и классов, с помощью которых реализуется требуемое поведение программной системы.

В основе такого преобразования лежат ВИ, созданные при определении требований к программной системе.

При этом рассматриваются только функциональные требования и игнорируются нефункциональные.

Проектирование – это уточнение результатов анализа, направленное на оптимизацию с учётом ограничений, накладываемых нефункциональными требованиями, средой реализации и т. д.

Анализ и проектирование

Роли

- **Системный архитектор** – руководит работами по анализу и проектированию программную систему. Он определяет общую структуру каждого архитектурного представления, декомпозицию представлений, группировку элементов и интерфейсы между группами.
- **Разработчик** – проектирует классы и отношения между ними Он определяет, как согласовывать классы со средой реализации.
- **Разработчик БД** – отвечает за проектирование базы данных программной системы.

Анализ и проектирование

Артефакты

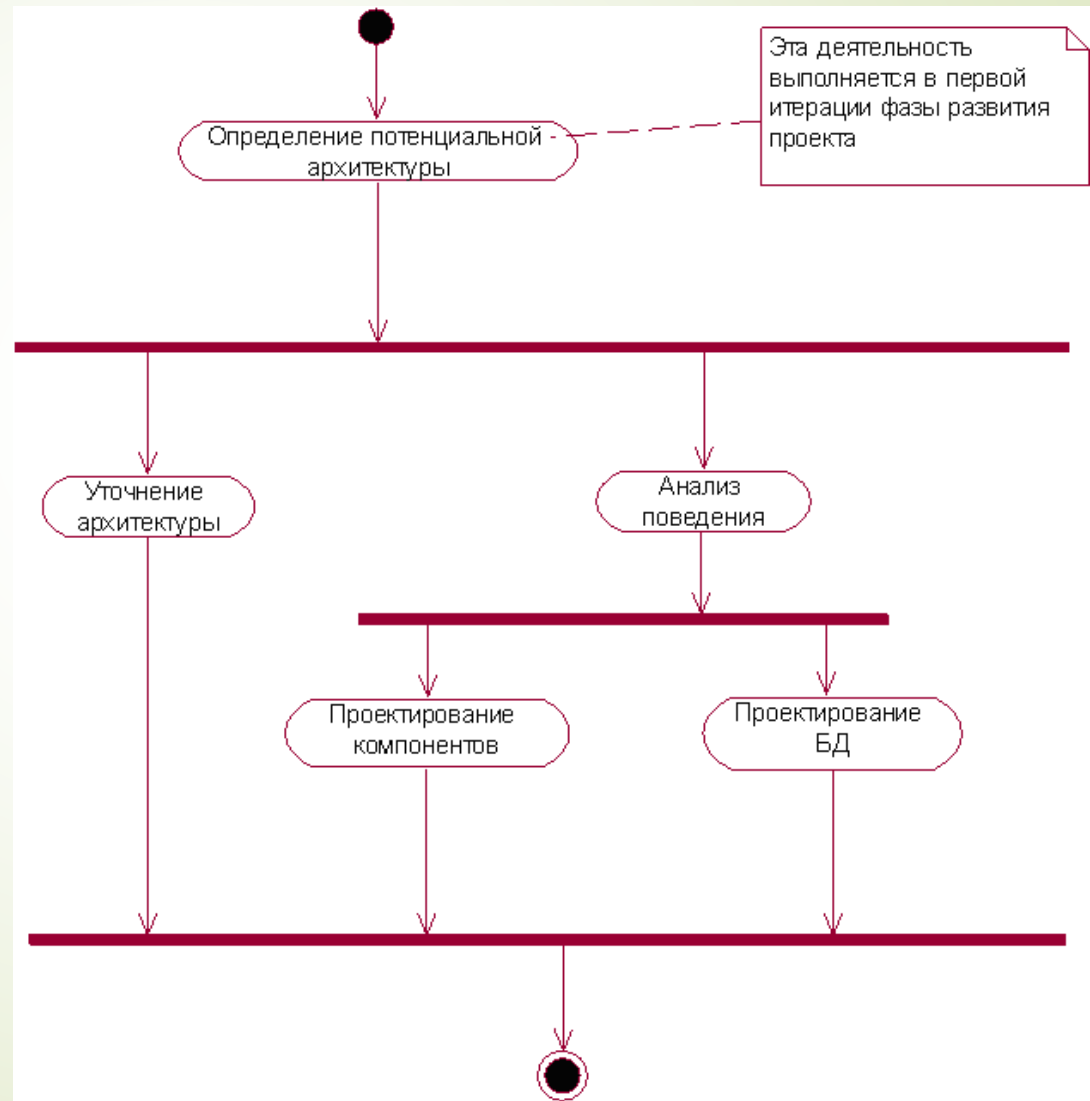
В процессе анализа и проектирования создаются следующие документы:

- **Модель проектирования** – это основная модель программной системы. Она описывает подсистемы, пакеты, компоненты, интерфейсы и классы, а также их взаимодействия, обеспечивающие требуемое поведение программной системы.
- **Документ «Архитектура программной системы»**, в котором собраны различные архитектурные представления программной системы.
- **Модель данных** – это описание структуры данных, хранимых в БД (например, реляционная модель данных).

Анализ и проектирование

Технологический процесс

- Упрощённая схема деятельности, выполняемых в отдельной итерации процесса анализа и проектирования (см. рис).
- Выполнение некоторых деятельности зависит от фазы разработки, что показано в виде комментариев на диаграмме деятельности.



Анализ и проектирование

Определение потенциальной архитектуры

Данная деятельность включает архитектурный анализ и анализ ВИ.

Определяется первоначальный набор архитектурно значимых элементов и механизмов реализации, выполняется начальное разбиение на уровни, определяется структура системы, выбираются ВИ, которые будут реализовываться в первой итерации фазы развития проекта. В результате создаётся эскиз архитектуры программной системы.

На основе анализа архитектурно значимых ВИ определяются основные классы, которые включаются в модель анализа.

В модель анализа включаются диаграммы, описывающие взаимодействие основных классов.

Анализ и проектирование

Уточнение архитектуры

Деятельность включает определение механизмов проектирования, элементов проекта, объединения существующих элементов проекта, описание архитектуры реального времени (если проектируемая программная система относится к этому классу). В результате выполнения этих работ достигаются следующие цели:

- Обеспечивается переход от анализа к проектированию путём определения из элементов и механизмов анализа элементов и механизмов проектирования.
- Поддерживается целостность и непротиворечивость архитектуры путём интеграции новых элементов проекта, определяемых в текущей итерации, с уже существующими и повторного использования доступных элементов проекта.
- Осуществляется плавный переход от проектирования к реализации.

Анализ и проектирование

Анализ поведения

Эта деятельность включает анализ ВИ, определение элементов проекта и обзор проекта.

Эта деятельность имеет целью преобразование описаний поведения в виде ВИ в набор элементов проекта (классы, отношения, операции и др.).

Анализ и проектирование

Проектирование компонентов

Цели данной деятельности состоят в:

- Определении и уточнении элементов проекта путём подробного описания того, как эти элементы реализуют требуемое поведение.
- Определении и уточнении реализации ВИ на основе новых элементов проекта.
- Контроле и рецензировании проекта по мере его развития.

Проектируются ВИ, подсистемы, классы и компоненты программной системы. Точно описываются интерфейсы компонентов и их реализация.

Анализ и проектирование

Проектирование БД. Данная деятельность выполняется для проектов, использующих базы данных. Она включает:

- Определение персистентных (постоянно хранимых) классов;
- Проектирование структуры БД для хранения таких классов;
- Определение механизмов и стратегий хранения и доступа к хранимым данным, удовлетворяющих требованиям к производительности и надёжности программной системы.

Анализ и проектирование связывает управление требованиями и реализацию. В этом технологическом процессе создаётся модель проектирования.

Одно из её представлений – логическая модель – отражает декомпозицию программной системы в набор логических элементов (классы, подсистемы, взаимодействия).

Процедурное представление отображает эти элементы в процессы и подпроцессы системы. Представление развёртывания отображает эти процессы в набор узлов вычислительного комплекса, на которых они выполняются.

Реализация

Цели процесса реализации

Основные цели процесса можно сформулировать следующим образом:

- Определить структуру кода в виде уровней.
- Реализовать компоненты, классы и объекты.
- Провести блочное тестирование компонент.
- Интегрировать разработки, выполненные отдельными разработчиками, в единую исполняемую систему.
- В процесс реализации не включено тестирование всей программной системы, для которого в RUP предусмотрен отдельный процесс.

Реализация

Особенности процесса реализации

- RUP предполагает поэлементную интеграцию в течение всего жизненного цикла. Это означает, что коды пишутся небольшими блоками, после чего они объединяются в единое целое путём постепенного добавления блоков. Это упрощает процесс локализации ошибок.
- Предусмотрено два уровня интеграции – интеграция результатов работы группы разработчиков в подсистему и интеграция подсистем в программную систему. Интеграция происходит в каждой итерации в соответствии с планом итерации, где определены ВИ, которые проектируются и реализуются в этой итерации.
- Таким образом, план итерации определяет классы, которые будут реализованы в этой итерации.

Реализация

Особенности процесса реализации

- В фазе конструирования создаётся эволюционный прототип системы, который со временем развивается в конечную программную систему.
- Это прототип используется для демонстрации фрагментов программной системы заказчику и руководству. По результатам представления прототипа можно получить замечания, которые позволяют уточнить, изменить или дополнить требования к программной системе.
- RUP декларирует возможность создания, помимо эволюционных, поведенческих одноразовых прототипов для проведения определённых исследований, касающихся функциональных возможностей системы.

Реализация

Особенности процесса реализации

- В RUP декларируется необходимость соответствия модели и программного кода.
- При этом допускается возможность изменения кода с последующей переработкой модели, которая обеспечивала бы требуемое соответствие.
- Для этой цели используют инструментальные средства, включающие возможности автоматического реинжиниринга.

Реализация

Роли

- **Конструктор (кодировщик)** разрабатывает компоненты и классы, выполняет блочное тестирование.
- **Системный интегратор** выполняет интеграцию элементов в программные конструкции (систему и подсистемы).
- **Архитектор** определяет структуру реализации (организацию уровней и подсистем).
- **Рецензент кода** проверяет качество программного кода и его соответствие стандартам проекта.

Реализация

Артефакты

- **Подсистема реализации** - это набор компонент и других подсистем реализации, используемых для образования модели реализации. Это понятие позволяет иерархически представить модель реализации.
- **Компонент** - часть программного кода.
- **План интеграции** - документ, определяющий порядок реализации компонентов и подсистем.

Реализация

Процесс

- **Создание модели реализации.** Эта деятельность выполняется в фазе развития. Целью её является построение модели реализации, которая позволит наилучшим образом выполнить разработку и кодирование. При этом конечный продукт будет создаваться посредством последовательно укрупняющихся прототипов.
- **Планирование итерации.** Для каждой итерации надо определить, какие подсистемы должны быть реализованы в этой итерации и порядок их интегрирования. За какую подсистему отвечает конкретный конструктор, определяющий порядок реализации классов.
- **Реализация компонентов.** Выполняется написание исходных кодов программ, их компиляция, формирование исполняемого кода и блочное тестирование. Блочное тестирование выполняет сам кодировщик. Это, по сути, автономная отладка разработанных программ. При обнаружении ошибок в исходный код вносятся изменения, после чего указанные действия повторяются. После завершения блочного тестирования проверяется качество исходного кода и его соответствие принятым стандартам.

Реализация

Процесс

- **Интеграция подсистем.** Если подсистема разрабатывается группой исполнителей, выполняется интеграция компонентов, разработанных всеми членами группы. Интеграция подсистемы завершается созданием набора программных конструкций подсистемы, каждая из которых тестируется по отдельности.
- **Интеграция системы.** Выполняется интеграция системы путём последовательного добавления в неё новых подсистем, созданных в рамках текущей итерации. Интеграция подсистемы завершается созданием набора программных конструкций подсистемы, каждая из которых тестируется по отдельности. Для тестирования системы в RUP предусмотрен отдельный процесс.

Тестирование

Тестирование – это процесс, позволяющий оценить качество производимого продукта.

Качественный программный продукт должен отвечать предъявляемым к нему требованиям как функциональным, так и нефункциональным. Программная система должна реализовывать все требуемые ВИ и не иметь дефектов – отличий реально существующих свойств или поведения от требуемых.

Кроме того, программная система должна обладать свойствами надёжности (должны отсутствовать зависания, аварийные отказы и пр.), безопасности, обеспечивать нужную производительность, быть удобной в эксплуатации, расширяемой и т. д. Таким образом, тестирование представляет собой процесс анализа программной системы, направленный на выявление дефектов и на оценку свойств программной системы.

Тестирование

Цели процесса тестирования

Целью тестирования является оценка качества программного продукта путём:

- Проверки взаимодействия компонентов.
- Проверки правильности интеграции компонентов.
- Проверки точности реализации всех требований и выявления дефектов.

Тестирование

Особенности процесса тестирования в RUP

- Тестирование – это итеративный процесс, выполняемый **во всех фазах жизненного цикла**.
- Работа над тестами начинается с самого начального этапа выявления требований к будущему продукту и тесно интегрируется с текущими задачами.
- На каждую итерацию определяется цель тестирования и методы её достижения.
- В конце каждой итерации определяется, насколько эта цель достигнута, нужны ли дополнительные испытания, не следует ли изменить принципы и инструменты тестирования.

Тестирование

Особенности процесса тестирования в RUP

- Каждый найденный дефект регистрируется в базе данных проекта с описанием ситуации, в которой он найден.
- Аналитик определяет, действительно ли это дефект и не является ли он повтором обнаруженного ранее дефекта.
- Найденному дефекту присваивается **приоритет**, определяющий важность исправления.
- Конструктор, ответственный за разработку подсистемы, компоненты или класса, или другой исполнитель, назначенный руководителем, приступает к устранению дефекта.
- Порядок исправления дефектов регулируется их приоритетами. Тестировщик повторяет выполнение тестов и убеждается (или не убеждается) в устранении дефекта.

Тестирование

Роли

- **Разработчик тестов** отвечает за планирование, разработку и реализацию тестов. Он создаёт план и модель тестирования, методики испытаний (см. далее) и выполняет оценку результатов тестирования.
- **Тестировщик (испытатель)** отвечает за выполнение системного тестирования. В его обязанности входит настройка и выполнение тестов, оценки выполнения теста, восстановление после ошибок, регистрация выявленных дефектов.

Тестирование

Артефакты

В процессе тестирования создаются следующие документы:

- **План тестирования** – документ, определяющий стратегию тестирования в каждой итерации.

Он содержит описание целей и задач тестирования в текущей итерации, а также стратегий, которые будут использоваться.

В плане указывается, какие потребуются ресурсы, и приводится перечень тестов.

Тестирование

Артефакты

В процессе тестирования создаются следующие документы:

- **Модель тестирования** – это представление того, что и как будет тестироваться. Модель включает набор контрольных задач, методик испытания, сценариев испытаний и ожидаемых результатов (test cases), тестовых скриптов и описаний взаимодействий тестов.
- **Контрольная задача** – набор тестовых данных, условий выполнения тестов и ожидаемых результатов.
- **Методика испытаний** – документ, содержащий указания по настройке и выполнению контрольных задач, а также по оценке получаемых результатов.
- **Сценарий тестирования** – это упрощённое описание теста, включая исходные данные, условия и последовательности выполнения действий и ожидаемые результаты.
- **Тестовый скрипт** является программой, выполняемой при автоматическом тестировании с помощью инструментальных средств тестирования.
- **Описание взаимодействия тестов** представляет собой диаграмму последовательностей или коопераций, отражающую упорядоченный во времени поток сообщений между компонентами тестов и объектом тестирования.

Тестирование

Артефакты

В процессе тестирования создаются следующие документы:

- **Результаты тестирования** и данные, полученные в процессе выполнения тестов.
- **Модель рабочей нагрузки** используется для моделирования внешних функций, выполняемых конечными пользователями, объёмов этих функций и нагрузки, создаваемой этими функциями. Модель предназначена для проведения нагрузочного и/или стрессового тестирования, имитирующего работу системы в реальных условиях.
- **Дефекты** – это описания обнаруженных при проведении тестирования фактов несоответствия системы предъявляемым требованиям. Они представляют собой вид запросов на внесение изменений.

Тестирование

Процесс

Работы по тестированию выполняются в каждой итерации во всех фазах, но цели и задачи в разных фазах проекта существенно различные.

- Фаза **вхождения в проект**. В этой фазе выполняется подготовка к тестированию. Она включает:
 - Создание плана тестирования, содержащего требования к тестам и стратегии тестирования. Может создаваться единый план для всех видов тестирования (функциональное, нагрузочное и т. д.) или отдельные планы для каждого вида.
 - Анализ объёма тестирования.
 - Формулирование критериев качества и завершения тестирования.
 - Установку и подготовки к работе инструментальных средств тестирования.
 - Формулирование требований к проекту разработки программной системы, определяемых потребностями тестирования.

Тестирование

Процесс

- **Фаза развития.** В итерациях этой фазы начинается построение модели тестирования и связанных с ней артефактов. Поскольку в этой фазе уже присутствует модель ВИ можно начинать проектировать сценарии тестирования. В то же время нецелесообразно выполнение тестов, поскольку обычно в этой фазе ещё не существует завершённых фрагментов программной системы. Выполняются следующие деятельности:
 - Создание плана тестирования для каждой итерации.
 - Разработка сценариев тестирования.
 - Создание заготовок тестовых скриптов.
 - Разработка контрольных задач.
 - Разработка методики испытаний.
 - Разработка модели рабочей нагрузки.

Тестирование

Процесс

- **Фаза конструирования.** В этой фазе появляются завершённые фрагменты систем и прототипы, которые должны тестироваться. При этом практически в каждой итерации проверяются все модули (как ранее разработанные и протестированные, так и новые, добавленные в текущей итерации). Тесты, применённые в предыдущих итерациях, используются и на последующих для регрессионного тестирования, то есть для проверки того, что ранее реализованная функциональность системы сохранилась в новой итерации. Выполняются следующие деятельности:
 - Создание плана тестирования для каждой итерации.
 - Уточнение и дополнение модели тестирования.
 - Выполнение тестов.
 - Описание обнаруженных дефектов.
 - Описание результатов тестирования.
 - Оценка результатов тестирования.

По результатам тестирования вносятся изменения в программный код с целью устранения выявленных дефектов, после чего тестирование выполняется повторно.

Тестирование

Процесс

- **Фаза развертывания.** В итерациях этой фазы выполняется тестирование всей программной системы как программного продукта. Выполняемые деятельности аналогичны деятельности предыдущей фазы. Выявление дефектов определяет необходимость внесения изменений и повторного тестирования. Итерационный процесс повторяется до тех пор, пока не будут выполнены критерии завершения тестирования.
- Оценка результатов тестирования производится на основе метрик тестирования, позволяющих определить качество тестируемой программной системы и самого процесса тестирования.

Тестирование

Инструментальная поддержка

- Поскольку итерационный процесс тестирования предусматривает многократное повторение тестов, ручное тестирование становится неэффективным и не позволяет тщательно оценить качество программного продукта.
- В особенности это касается нагрузочного и стрессового тестирования, где требуется моделировать рабочую нагрузку и накапливается значительный объем данных.
- Выход состоит в применении инструментальных средств, поддерживающих автоматизацию составления и выполнения тестов.

Процессы поддержки

RUP предусматривает три процесса поддержки:

- Управление проектом;
- Управление конфигурацией;
- Управление средой.

Процессы поддержки

Управление проектом. Цели

Основной целью процесса управления проектом является обеспечение руководства проектом, направленное на успешную сдачу программного продукта. RUP акцентирует внимание на планировании жизненного цикла и отдельных итераций, управлении рисками, наблюдаемости хода проекта и метриках проекта.

Планирование предполагает созданий двух видов планов:

- План фаз – крупномасштабный план проекта, показывающий основные вехи жизненного цикла (даты завершения больших этапов – фаз, выпуска эволюционных прототипов и т. д.) и требуемые ресурсы. Он создаётся в начале фазы исследования и обновляется по мере необходимости.
- План итераций создаётся для каждой итерации и предназначен для определения и распределения задач между участниками проекта.

Процессы поддержки

- **Риском** будем называть все то, что может стоять на пути к успеху проекта и на данный момент является неизвестным или неопределённым.

Главная идея управления рисками заключается в том, что не нужно пассивно ждать, пока риск станет проблемой, но требуется заблаговременно определять линию поведения. Управление рисками означает определение и оценку рисков, принятие линии поведения, направленной на устранение, снижение вероятности риска, а также выбор действий на случай реализации риска.

- Для контроля и управления проектом используются **измерения**. Они проводятся для того чтобы установить, насколько отдалено текущее состояние проекта от поставленной цели, спланировать работы и определить, как можно повысить эффективность процесса.

Процессы поддержки

Управление проектом. Деятельности

- **Открытие нового проекта.** Эта деятельность выполняется только в первой итерации. Осуществляется инициализация проекта, определяются и оцениваются риски, разрабатывается бизнес-план. Цель – перевод проекта в стадию, когда возможно принятие решения о продолжении или об отказе от проекта.
- **Оценка области действия проекта и рисков.** Целью является пересмотр и уточнение возможностей и характеристик проекта, а также связанных с ним рисков.
- **Создание плана разработки программной системы.** Создаётся план разработки программной системы, включающий перечень рисков, планы измерений, управления рисками, разрешения проблем, принятия продукта. Определяется структура и ресурсы проекта. Разрабатывается план фаз проекта.

Процессы поддержки

Управление проектом. Деятельности

- **Планирование итерации.** Разрабатывается план очередной итерации, уточняется и корректируется бизнес-план и план разработки программной системы. План итерации подробно описывает, что должно произойти за время итерации. Он определяет исполнителей и выполняемые ими работы. При создании плана итерации необходимо:
 - Сформулировать объективные критерии успеха итерации. Они будут использоваться при её оценке;
 - Определить конкретные, измеримые артефакты, которые требуется разработать или изменить, а также выполняемые для этого работы;
 - Использовать типичную итерационную декомпозицию работ для реальных действий, которые должны быть произведены;
 - Использовать смету при определении продолжительности и объёма работ для каждого вида деятельности, удерживая все значения в пределах бюджета.

Процессы поддержки

Управление проектом. Деятельности

- **Наблюдение и контроль.** Эта деятельность включает распределение работ и создание графика работ, наблюдение за состоянием проекта, обработку исключительных ситуаций, и создание отчёта о состоянии.

Выполняется обработка предложенных изменений и включение их в график работ текущей или последующих итераций, производится наблюдение за активными рисками, даётся оценка прогресса и качества проекта.

В RUP постоянно выполняемые оценки состояния проекта являются основой для решения разных проблем и управления рисками проекта. Если команда разработчиков выявила проблему, назначается сотрудник, ответственный за её разрешение, и определяется дата, когда проблема должна быть разрешена.

Ход процесса должен регулярно контролироваться, а обновления должны выполняться по мере необходимости.

Процессы поддержки

Управление проектом. Деятельности

- **Управление итерацией.** Целью этой деятельности является получение достаточных для выполнения итерации ресурсов, разделение требуемых работ, оценка результатов итерации.
- **Завершение фазы.** Выполняются работы, завершающие выполнение фазы. Даются ответы на следующие основные вопросы:
 - Решены ли все основные проблемы предыдущей итерации?
 - Известно ли состояние всех основных артефактов (см. ниже управление конфигурацией)?
 - Рассмотрены ли все проблемы развёртывания?
 - При удовлетворительном состоянии проекта выдаётся разрешение на переход к следующей фазе.
- **Завершение проекта.** Руководитель проекта подготавливает проект к завершению. Готовится заключительная оценка состояния. При успешной сдаче проекта заказчик получает программный продукт в пользование. Высвобождающие ресурсы могут быть перераспределены (использованы в других проектах).

Процессы поддержки

Управление конфигурацией и изменениями. Цели

- Целью управления конфигурацией и изменениями является поддержание целостности артефактов с учётом возможности внесения в них изменений в соответствии с запросами на изменения.
- Этот вспомогательный процесс распространяется на весь жизненный цикл программного продукта и складывается из трёх отдельных процессов.

Процессы поддержки

Управление конфигурацией и изменениями. Цели

Управление конфигурацией (Configuration management).

- Процесс управления конфигурацией (УК) связан с определением артефактов, зависимостей между ними и конфигураций, являющихся непротиворечивыми множествами взаимосвязанных артефактов. Сюда же относятся вопросы распределения рабочих сред между участниками проекта с целью предотвращения ненужного дублирования документов.
- УК декларирует, что все артефакты подчиняются **версионному управлению**. По мере развития проекта у каждого артефакта появляются множество **версий**. Все они сохраняются в **архиве проекта**, причём ведётся история изменений, в которой указывается причины и цели создания каждой версии каждого артефакта.
- Записанную в архив проекта версию нельзя изменить, можно лишь создать новую версию. Кроме того, знания о зависимостях между артефактами позволяют точно повторять действия при создании наборов артефактов (например, при сборке программы из отдельных файлов). Создание версий и связей поддерживается с помощью инструментальных средств УК.

Процессы поддержки

Управление конфигурацией и изменениями. Цели

УК позволяет:

- исключить возможность потери артефактов,
- обеспечить возможность «возврата назад» в случаях, когда принятые решения и полученные при их реализации версии артефактов не устраивают заказчика или разработчиков проекта,
- гарантировать точное повторение действий над группой связанных артефактов в итерациях,
- избегать дублирования артефактов и связанной с этим возможности привнесения дополнительных дефектов,
- поддерживать наблюдаемость хода выполнения работ по проекту путём предоставления нужной информации.

Процессы поддержки

Управление конфигурацией и изменениями. Цели

Управление внесением изменений

Деятельности этого процесса направлены на сбор и сохранение **запросов на внесение изменений**, поставляемых как участниками разработки программной системы, так и внешними сторонами. Запросы на внесение изменений могут появляться по разным причинам (исправление дефекта, улучшение параметра качества продукта, например, производительности, задание дополнительных требований и т. д.)

Каждый запрос на внесение изменений сохраняется в **БД** проекта и может находиться в разных **состояниях** в зависимости от хода выполнения работ по этому запросу (новый, зарегистрированный, принятый, завершённый и др.). Состояния запросов изменяются участниками проекта в соответствии с выделенными правами.

По мере работы с запросом в него добавляется информация (например, сначала это может быть только описание дефекта, затем добавляются результаты анализа, указываются затрагиваемые артефакты, назначается исполнитель). С запросом может быть связан **приоритет**, позволяющий определить порядок выполнения работ по запросам на изменения.

Процессы поддержки

Управление конфигурацией и изменениями. Цели

Управление состоянием и измерениями

Этот процесс связан с управлением проектом и направлен на предоставление информации, полезной для оценки:

- Состояния, прогресса, общих тенденций и качества продукта.
- Издержек и расходов.
- Проблемных областей, требующих внимания.
- Того, что сделано и что необходимо сделать.

Вся необходимая информация находится в базе данных и архиве проекта. Руководитель работ оценивает состояние дел путём непосредственного просмотра запросов, истории версий артефактов или на основе анализа различных отчётов, формируемых инструментальным средством УК. Анализ позволяет руководителю оценить имеющиеся ресурсы и принять необходимые управленческие решения.

Процессы поддержки

Деятельности

- ▶ **Планирование конфигурации проекта и управления изменениями.** Описываются все виды деятельности, связанные с УК, которые надо выполнить. Описывается процесс контроля над изменениями.
- ▶ **Создание среды УК.** Целью этой деятельности является создание рабочей среды, в которой будут доступны все артефакты, будет обеспечена разработка, интеграция и архивирование продукта.
- ▶ **Создание рабочей среды исполнителей.** В пределах своей рабочей среды исполнитель получает доступ к артефактам проекта, имеет возможность вносить в них изменения и предлагать внесение изменений в проект в целом. Изменения, сделанные отдельными исполнителями, направляются в рабочую среду интеграции.

Процессы поддержки

Деятельности

- **Управление версиями.** При записи артефакта в архив проекта формируется его новая версия. Управление версиями предусматривает обязательное указание причин и целей создания версии. При выборе артефакта из архива можно указать конкретную версию.
- **Управление запросами на внесение изменений.** Целью этой деятельности является обеспечение последовательного внесения изменений в проект и информирование о состоянии продукта, его изменениях и о влиянии изменений на стоимость и сроки выполнения проекта.
- **Наблюдение за состоянием конфигурации.** Наблюдаемость процесса разработки обеспечивается путём доступа руководства проекта к хранимым артефактам и запросам на изменения, а также путём предоставления отчётов о состоянии конфигурации. При этом инструментальные средства, поддерживающие управление конфигурацией и изменениями, могут выполнять требуемые измерения. Например, можно показать процент завершённых запросов, число открытых запросов высокого приоритета и т. д.

Процессы поддержки

Управление средой. Цели

Многие виды деятельности, предусмотренные RUP, могут быть автоматизированы посредством инструментальных средств, что позволит избежать наиболее трудоёмких, напряжённых и подверженных ошибкам аспектов разработки программной системы.

Цель процесса управления средой – процедурная и инструментальная поддержка организации, разрабатывающей программной системы. Она включает:

- Выбор и приобретение инструментальных средств.
- Настройку инструментальных средств под требования организации-разработчика.
- Конфигурирование процесса.
- Усовершенствование процесса.
- Создание технических служб поддержки.

Процессы поддержки

Роли и артефакты

- Основным исполнителем процесса является **технолог**. В его обязанности входит конфигурирование процесса перед запуском проекта и усовершенствование процесса в ходе проектных работ. Поддержка аппаратной и программной среды разработки ложится на плечи **системного администратора**.
- Главным создаваемым артефактом является **план разработки**, описывающий процесс, используемый в данном проекте. В нем указывается, какие артефакты, предусмотримые в RUP, будут использоваться в проекте и каким образом.

Процессы поддержки

Деятельности

- **Подготовка среды к реализации проекта.** Выполняется оценка текущего состояния организации-разработчика и имеющейся инструментальной поддержки. Создаётся предварительный план разработки. Составляется перечень инструментальных средств, которые предполагается использовать при разработке. Составляется перечень шаблонов для производства основных артефактов.
- **Подготовка среды к итерации.** Производится дополнение и уточнение плана разработки, выполняется подготовка и настройка инструментальных средств, которые будут использоваться в итерации. Составляется набор шаблонов для артефактов, создаваемых в итерации. Осуществляется подготовка сотрудников к использованию инструментальных средств.
- **Подготовка руководящих директив.** Для каждого основного технологического процесса подготавливается набор директив на основе анализа проблем и дефектов предыдущих итераций.