# Тема 3
# МОДЕЛИ И МЕТОДОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Процесс жизни любой системы или программного продукта может быть описан посредством модели жизненного цикла, состоящей из стадий. Модели могут использоваться для представления всего жизненного цикла от замысла до прекращения применения или для представления части жизненного цикла, соответствующей текущему проекту. Модель жизненного цикла представляется в виде последовательности стадий, которые могут перекрываться и (или) повторяться циклически в соответствии с областью применения, размером, сложностью, потребностью в изменениях и возможностях. Каждая стадия описывается формулировкой цели и выходов. Процессы и действия жизненного цикла отбираются и исполняются на этих стадиях для полного удовлетворения цели и результатам каждой стадии. Различные организации могут использовать различные стадии в пределах жизненного цикла. Однако каждая стадия реализуется организацией, ответственной за эту стадию, с надлежащим рассмотрением информации, имеющейся в планах жизненного цикла и решениях, принятых на предшествующих стадиях. Аналогичным образом организация, ответственная за текущую стадию, ведет записи принятых решений и записи допущений, относящихся к последующим стадиям данного жизненного цикла [2, 25].

## 3.1. Модели жизненного цикла программного обеспечения

Под моделью жизненного цикла ПО понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении ЖЦ. Модель ЖЦ зависит от спецификации, масштаба и сложности проекта и спецификации условий, в которых система создается и функционирует. Модель ЖЦ ПО включает в себя: стадии, результаты выполнения работ на каждой стадии, ключевые события – точки завершения работ и принятия решений. Модель ЖЦ любого конкретного ПО определяет характер процесса его создания, который представляет собой совокупность упорядоченных во времени, взаимосвязанных и объединенных в стадии работ, выполнение которых необходимо и достаточно для создания ПО, соответствующего заданным требованиям. Под стадией понимается часть процесса создания ПО, ограниченная определенными временными рамками и

заканчивающаяся выпуском конкретного продукта (моделей, программных компонентов, документации), определяемого заданными для данной стадии требованиями. На каждой стадии могут выполнятся несколько процессов, определённых в стандарте ГОСТ Р ИСО/МЭК 12207-2010, и наоборот один и тот процесс может выполняться на различных стадиях. Соотношение между стадиями и процессами также определяется используемой моделью ЖЦ ПО. Далее рассмотрим модели и их классификации [2, 25].

### 3.1.1. Каскадная модель

Первой моделью, получившей широкую известность и действительно структурирующей процесс разработки, является каскадная (водопадная) модель. Каждая стадия каскадной модели заканчивается получением некоторых результатов, которые служат в качестве исходных данных для следующей стадии. Требования к разрабатываемому ПО, определенные на стадии формирования требований, строго документируются в виде технического задания и фиксируются на все время разработки проекта.

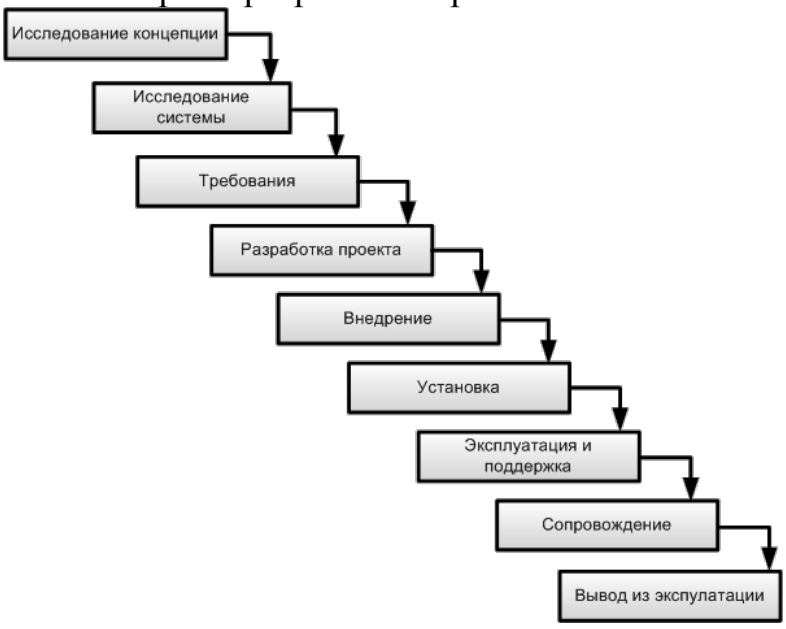


Рис. 3.1 Стандартная водопадная модель

Преимущества применения каскадной модели заключаются в следующем:

- на каждой стадии формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;

– выполняемые в логичной последователь стадии работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадная модель может использоваться при создании ПО, для которого в самом начал разработки можно достаточно точно и полно сформулировать все требования. В то же время этот подход обладает рядом недостатков, вызванных прежде всего тем, чьл реальный процесс создания ПО никогда полностью не укладывался в такую жесткую схему.

Спустя непродолжительное время после появления на свет каскадная модель была доработана Уинстом Ройсом с учетом взаимозависимости этапов и необходимости возврата на предыдущие ступени, что может быть вызвано, например, неполнотой требований или ошибками в формировании задания. Процесс создания ПО носит как правило, итерационный характер: результаты очередной стадии часто вызывают изменения в проектных решениях, выработанных на более ранних стадиях. Таким образом, постоянно возникает потребность в возврате к предыдущим стадиям и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ПО принимает вид, изображенный на рисунке *.1*.

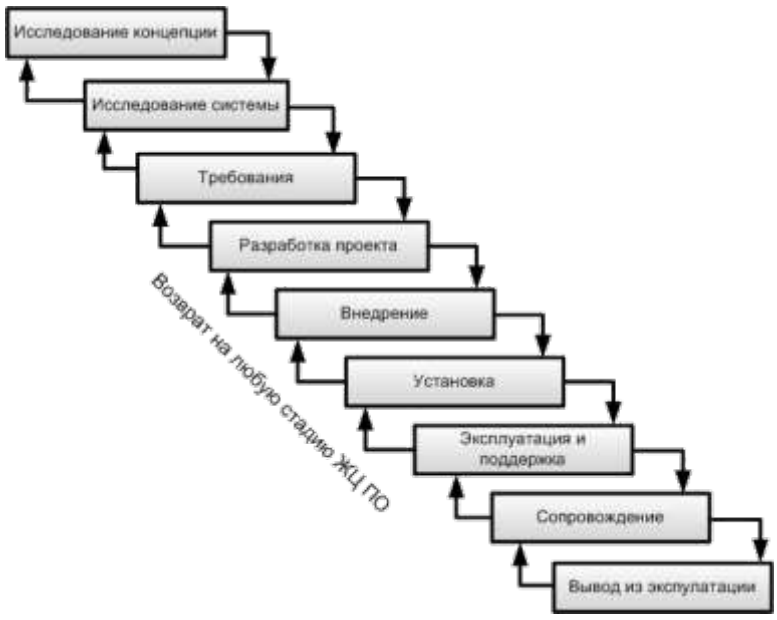


Рис. 3.1. Модифицированная водопадная модель

Наиболее распространенным результатом каскадного похода к разработке ПО является поздняя неудача. Кажется, что проекты выполняются нормально, но только до тех пор, пока работы не вступят в

завершающий этап, и тогда выясняется, что потребители недовольны созданным продуктом [25].

### 3.1.2. V-образная модель, как разновидность каскадной модели

Основной принцип V-образной модели заключается в том, что детализация проекта возрастает при движении слева направо, одновременно с течением времени, и ни то, ни другое не может повернуть вспять. Итерации в проекте производятся по горизонтали, между левой и правой сторонами буквы.

V-модель – вариация каскадной модели, в которой задачи разработки идут сверху вниз по левой стороне буквы V, а задачи тестирования – вверх по правой стороне буквы V. Внутри V проводятся горизонтальные линии, показывающие, как результаты каждой из стадий разработки влияют на развитие системы тестирования на каждой из стадий тестирования. Модель базируется на том, что приемо-сдаточные испытания основываются, прежде всего, на требованиях, системное тестирование – на требованиях и архитектуре, комплексное тестирование – на требованиях, архитектуре и интерфейсах, а компонентное тестирование – на требованиях, архитектуре, интерфейсах и алгоритмах.

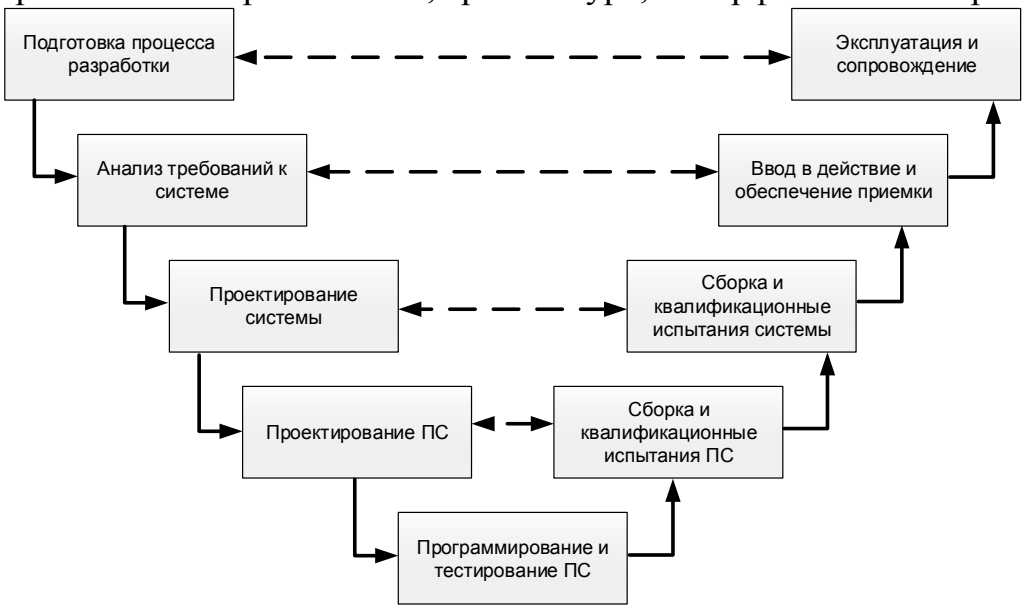


Рис. 3.2 – V-образная модель

Особенностью данной модели является разбиение стадий на три логических этапа: проектирование (детализация требований), реализация, тестирование.

V-модель дает организациям и проектным группам руководство по выполнению и завершению проектов последовательным и воспроизводимым образом. Применение принципов V-модели гарантирует выявление и фиксацию требований пользователей. Утвержденные требования могут быть переведены в функции готового приложения, (и) приложение отражает требования пользователей [2, 25].

### 3.1.3. Итеративный инкрементный подход к разработке (эволюционная модель)

#### 3.1.3.1 Итеративная модель

Итеративная модель предполагает разбиение жизненного цикла проекта на последовательность итераций, каждая из которых напоминает «мини-проект», включая все фазы жизненного цикла в применении к созданию меньших фрагментов функциональности, по сравнению с проектом, в целом. Цель каждой итерации – получение работающей версии программной системы, включающей функциональность, определенную интегрированным содержанием всех предыдущих и текущей итерации. Результата финальной итерации содержит всю требуемую функциональность продукта. Таким образом, с завершением каждой итерации, продукт развивается инкрементально.

Шансы успешного создания сложной системы будут максимальными, если она реализуется в серии небольших шагов и если каждый шаг заключает в себе четко определенный результат, а также возможность возврата к результатам предыдущей успешной итерации, в случае неудачи. Перед тем, как пустить в дело все ресурсы, предназначенные для создания ПО, разработчик имеет возможность получать обратную связь из реального мира (заказчиков, пользователей) и исправлять возможные ошибки в проекте.

*Итеративная модель подразумевает возможность не только сборки работающей (с точки зрения результатов тестирования) версии системы* ***- прототипа****, но и её развертывания в реальных операционных условиях с анализом откликов пользователей для определения содержания и планирования следующей итерации.* Поскольку на каждом шаге мы имеем работающую систему, то можно:

- очень рано начать тестирование пользователями;
- принять стратегию разработки в соответствии с бюджетом, полностью защищающую от перерасхода времени или средств (в частности, за счет сокращения второстепенной функциональности).

### 3.1.3.2 Инкрементная модель

Идея, лежащая в основе инкрементной модели, состоит в том, что программную систему следует разрабатывать по принципу приращений, так, чтобы разработчик мог использовать данные, полученные при разработке более ранних версий ПО. Новые данные получаются как в ходе разработки ПО, так и в ходе его использования, где это возможно. Ключевые этапы этого процесса – простая реализация подмножества требований к программе и совершенствование модели в серии последовательных релизов до тех пор, пока не будет реализовано ПО во всей полноте. В ходе каждой итерации организация модели изменяется, и к ней добавляются новые функциональные возможности.

Для организации инкрементной разработки обычно выбирается характерный временной интервал, например, неделя. Затем в течение этого интервала происходит обновление проекта: добавляется новая документация как текстовая, так и графическая, расширяется набор тестов, добавляются новые программные коды и т. д. Теоретически шаги разработки могут выполняться и параллельно, но такой процесс очень сложно скоординировать. Инкрементная разработка проходит лучше всего, если следующая итерация начинается после того, как обновление всех артефактов в предыдущей итерации закончено, и существенно хуже, если время, требуемое на обновление артефактов, значительно превышает выбранный интервал [27].

*В результате каждой итерации получается работающее, но не полнофункциональное ПО, которое еще не является программным продуктом и не подлежит распространению. В результате каждой итерации создается версия некоторой части ПО.* Необходимо заметить, но как правило на каждой итерации определяются и реализуются новые требования, некоторые итерации могут быть целиком посвящены усовершенствованию существующей программы, например, с целью повышения ее производительности.

**Вывод**

С точки зрения структуры жизненного цикла эволюционную модель называют итеративной. С точки зрения развития продукта – инкрементальной. **Опыт показывает, что невозможно рассматривать каждый из этих взглядов изолировано.** Чаще всего такую смешанную эволюционную модель называют просто итеративной (говоря о процессе) и/или инкрементальной (говоря о наращивании функциональности продукта). Значимость эволюционной модели на основе организации итераций особо проявляется в снижении

неопределенности с завершением каждой итерации. В свою очередь, снижение неопределенности позволяет уменьшить риски. Рисунок 3.3 иллюстрирует идею эволюционной модели, предполагая, что итеративному разбиению может быть подвержен не только жизненный цикл в целом, включающий перекрывающиеся стадии – формирование требований, проектирование, конструирование и т.п., но и каждая стадия может, в свою очередь, разбиваться на уточняющие итерации, связанные, например, с детализацией структуры декомпозиции проекта – например, архитектуры модулей системы [25].

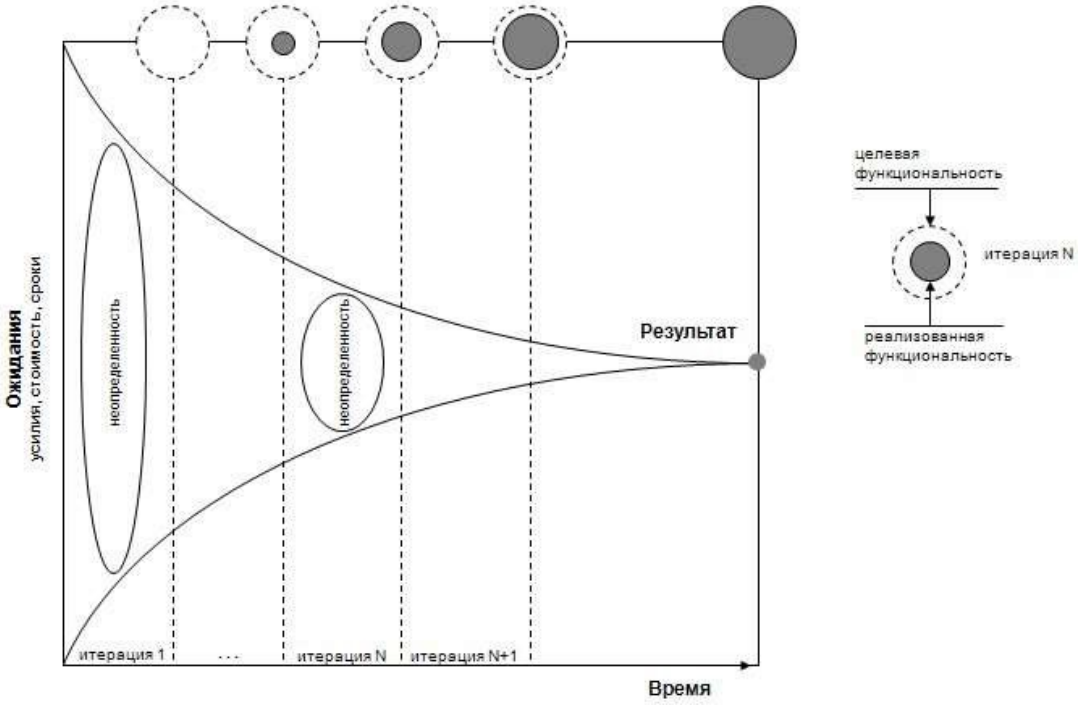


Рисунок 3.3 – Снижение неопределенности и инкрементальное расширение функциональности при итеративной организация жизненного цикла.

### 3.1.4. Спиральная модель, как разновидность эволюционной модели

В середине 1980-х годов Барри Боэм предложил свой вариант итерационной модели итеративной модели под названием «Спиральная модель». При использовании спиральной модели прикладное ПО создается в несколько итераций (витков спирали) методом прототипирования. Создание прототипов осуществляется в несколько итераций, или витков спирали. Каждая итерация соответствует созданию фрагмента или версии ПО, на ней уточняются цели и характеристики проекта, оценивается качество полученных результатов и планируются

работы следующей итерации. На каждой итерации производится тщательная оценка риска превышения сроков и стоимости проекта, чтобы определить необходимость выполнения еще одной итерации, степень полноты и точности понимания требований к системе, а также целесообразность прекращения проекта [2, 27].

Отличительной особенностью этой модели является специальное внимание рискам, влияющим на организацию жизненного цикла. Боэм формулирует 10 наиболее распространенных (по приоритетам) рисков:

- дефицит специалистов;
- нереалистичные сроки и бюджет;
- реализация несоответствующей функциональности;
- разработка неправильного пользовательского интерфейса;
- «золотая сервировка», перфекционизм, ненужная оптимизация и оттачивание деталей;
- непрекращающийся поток изменений;
- нехватка информации о внешних компонентах, определяющих окружение системы или вовлеченных в интеграцию;
- недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами;
- недостаточная производительность получаемой системы;
- «разрыв» в квалификации специалистов разных областей знаний.

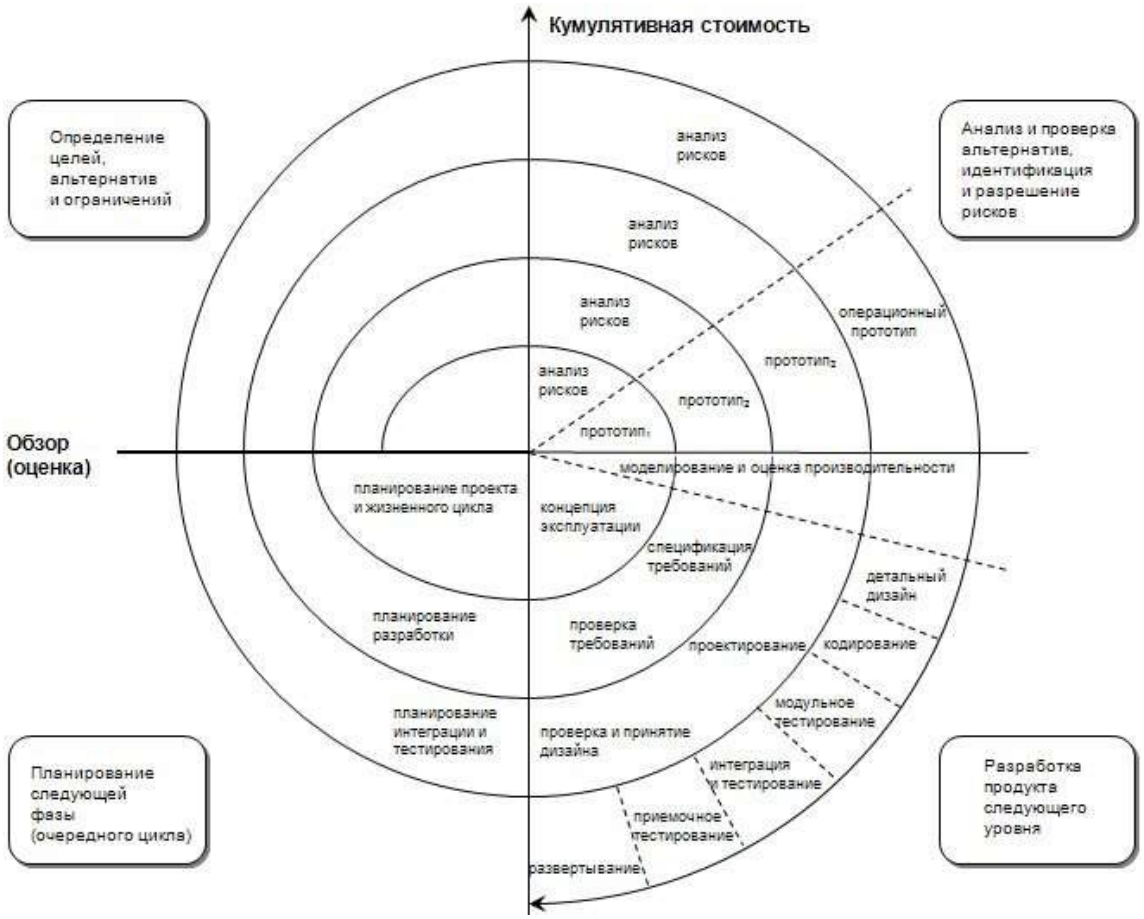


Рис. 3.4 Оригинальная спиральная модель жизненного цикла разработки по Боэму

Основная проблема спирального цикла – определение момента перехода на следующую стадию. Для её решения необходимо ввести временные ограничения на каждую из стадий ЖЦ. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статических данных, полученных в предыдущих проектах, и личного опыта разработчика.

Достоинствами спиральной модели являются: ускорение разработки (ранее получение результата за счет прототипирования), постоянное участие заказчика в процессе разработки, разбиение большого объема работы на небольшие части, снижение риска [2, 25].

**Заключение**

Естественное развитие каскадной и эволюционной модели привело к их сближению и появлению современных подходов - методологий, которые по существу представляет собой рациональное сочетание вышеописанных моделей. Различные варианты эволюционного подхода реализованы в большинстве современных технологий и методов: Rational Unified Process (RUP), Microsoft Solutions Framework (MSF), Agile и другие.

## 3.2. Методологии разработки ПО

Методологии представляют собой ядро теории управления разработкой ПО. К существующей классификации в зависимости от используемой в ней модели жизненного цикла (каскадные и эволюционные) добавилась более общая классификация на прогнозируемые и адаптивные методологии.

Прогнозируемые (предикативные) методологии фокусируются на детальном планировании будущего. Известны запланированные задачи и ресурсы на весь срок проекта. Команда с трудом реагирует на возможные изменения. План оптимизирован исходя из состава работ и существующих требований. Изменение требований может привести к существенному изменению плана, а также дизайна проекта.

Адаптивные (гибкие) методологии нацелены на преодоление ожидаемой неполноты требований и их постоянного изменения. Когда меняются требования, команда разработчиков тоже меняется. Команда, участвующая в адаптивной разработке, с трудом может предсказать будущее проекта. Существует точный план лишь на ближайшее время. Более удаленные во времени планы существуют лишь как декларации о целях проекта, ожидаемых затратах и результатах. Среди адаптивных методологий: (Scrum, Crystal, Extreme Programming, Adaptive Software Development, DSDM, Feature Driven Development, Lean software development). Рассмотрим самые основные и популярные методологии [26].

### 3.2.1. RUP (Rational Unified Process)

Один из самых известных процессов, использующих итеративную модель разработки – RUP. Он был создан во второй половине 1990-х годов в компании Rational Software. Термином RUP обозначает как методологию, так и продукт компании IBM (ранее Rational) для управления процессом разработки. Методология RUP описывает абстрактный общий процесс, на основе которого организация или

проектная команда должна создать специализированный процесс, ориентированный на ее потребности.

Основные характеристики:

- разработка требований, для описания требований в RUP используются прецеденты использования (use cases). Полный набор прецедентов использования системы вместе с логическими отношениями между ними называется моделью прецедентов использования. Каждый прецедент использования – это описание сценариев взаимодействия пользователя с системой, полностью выполняющего конкретную пользовательскую задачу. Согласно RUP все функциональные требования должны быть представлены в виде прецедентов использования.
- итеративная разработка, проект RUP состоит из последовательности итераций с рекомендованной продолжительностью от 2 до 6 недель. Перед началом очередной итерации определяется набор прецедентов использования, которые будут реализованы к её завершению.

#### 3.2.1.1 Архитектура

Можно сказать, что RUP – ориентированная на архитектуру методология. Считается, что реализация и тестирование архитектуры системы должны начинаться на самых ранних стадиях проекта. RUP использует понятие исполняемой архитектуры (executable architecture) – основы приложения, позволяющей реализовать архитектурно значимые прецеденты использования. Основы исполняемой архитектуры должны быть реализованы как можно раньше. Это позволяет оценить адекватность принятых архитектурных решений и внести необходимые коррективы еще в начале проекта. Таким образом, для первых нескольких итераций необходимо выбирать прецеденты, которые требуют реализации большей части архитектурных компонентов.

RUP поощряет использование визуальных средств для анализа и проектирования. Как правило, используется нотация и, соответственно, средства моделирования UML (такие как Rational Rose). Модель предметной области документируется в виде диаграммы классов, модель прецедентов использования – при помощи диаграммы прецедентов, взаимодействие компонентов системы между собой описывается диаграммой последовательности и т д.

#### 3.2.1.2 Жизненный цикл проекта

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

**Начало (Inception)**

Стадия «начало» обычно состоит из одной итерации. В ходе выполнения этой стадии необходимо:

- определить видение и границы проекта;
- создать экономическое обоснование;
- идентифицировать большую часть прецедентов использования и подробно описать несколько ключевых прецедентов;
- найти хотя бы одно возможное архитектурное решение;
- оценить бюджет, график и риски проекта.

Если после завершения первой итерации заинтересованные лица приходят к выводу о целесообразности выполнения проекта, проект переходит в следующую стадию. В противном случае проект может быть отменен или проведена еще одна итерация стадия «начало».

**Проектирование (Elaboration)**

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

- детальное описание большей части прецедентов использования;
- создание оттестированной (при помощи архитектурно значимых прецедентов использования) базовой архитектуры;
- снижение основных рисков и уточнение бюджета и графика проекта.

В отличие от каскадной модели, основным результатом этой стадии является не множество документов со спецификациями, а действующая система с 20-30% реализованных прецедентов использования [25].

**Построение (Construction)**

В этой стадии (длящейся от двух до четырех итераций) происходит разработка окончательного продукта. Вовремя ее выполнения создается основная часть исходного кода системы и выпускаются промежуточные демонстрационные прототипы.

**Внедрение (Transition)**

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

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

**Рабочий процесс**

В терминах RUP участники проектной команды создают так называемые артефакты (work products), выполняя задачи (tasks) в рамках определенных ролей (roles). Артефактами являются спецификации, модели, исходный код и т.п. Задачи разделяются по девяти процессным областям, называемым дисциплинами (discipline). В RUP определены шесть инженерных и три вспомогательные дисциплины. В них входят:

- **Бизнес-моделирование (Business Modeling**) – исследование и описание существующих бизнес-процессов заказчика, а также поиск их возможных улучшений.
- **Управление требованиями (Requirements Management)** – определение границ проекта, разработка функционального дизайна будущей системы и его согласование с заказчиком.
- **Анализ и проектирование (Analysis and Design)** – проектирование архитектуры системы на основе функциональных требований и ее развитие на протяжении всего проекта.
- **Реализация (Implementation)** – разработка, юнит-тестирование и интеграция компонентов системы.
- **Тестирование (Test)** – поиск и отслеживание дефектов в системе, проверка корректности реализации требований.
- **Развертывание (Deployment)** – создание дистрибутива, установка системы, обучение пользователей.
- **Управление конфигурациями и изменениями (Configuration and Change Management)** – управление версиями исходного кода и документации, процесс обработки запросов на изменение (Change requests).
- **Управление проектом (Project Management)** – создание проектной команды, планирование фаз и итераций, управление бюджетом и рисками.

– **Среда (Environment)** – создание инфраструктуры для выполнения проекта, включая организацию и настройку процесса разработки.

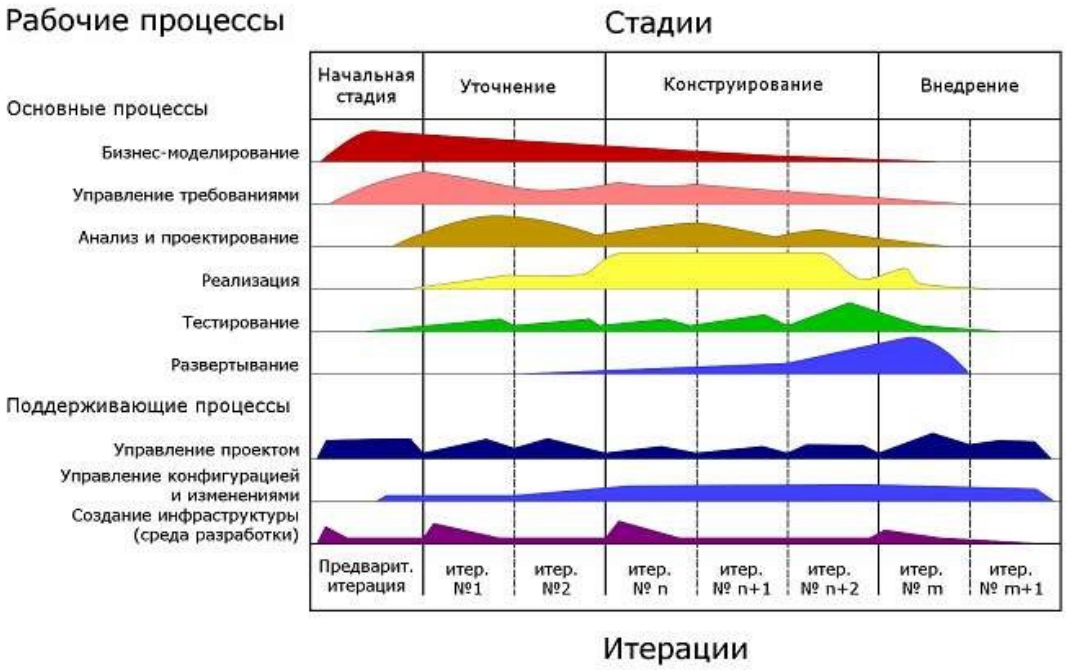


Рис. 3.5 Распределение усилий при выполнении проекта

**Заключение**

В ходе жизненного цикла проекта распределение усилий проектной команды между дисциплинами постоянно меняется. Например, как правило, в начале проекта большая часть усилий затрачивается на анализ и дизайн, а ближе к завершению – на реализацию и тестирование системы. Однако в общем случае задачи из всех девяти дисциплин выполняются параллельно.

Для полноценного внедрения RUP организация должна затратить значительные средства на обучение сотрудников. При этом попытка обойтись своими силами скорее всего будет обречена на неудачу – необходимо искать специалиста по процессам (process engineer) с соответствующим опытом или привлекать консультантов.

### 3.2.2. Microsoft Solutions Framework (MSF)

Данная методология описывает подход и организацию работы при создании программных продуктов. Подробно про методологию MSF вы

можете прочитать в переводе Microsoft Solutions Frameworks for Agile Software Development, которая входит в поставку Microsoft Team Foundation Server [23].

### 3.2.3. Scrum

Scrum предоставляет эмпирический подход к разработке ПО. Этот процесс быстр, адаптивен, умеет подстраиваться и отличен от каскадной модели. Scrum основан на повторяющихся циклах, это делает его более гибким и предсказуемым.

Для начала определим роли, которые участвуют в процессе: Scrum мастер (Scrum Master), Владелец продукта (Product Owner), Команда (Team).

**Scrum Мастер** - самая важная роль в методологии. Scrum Мастер отвечает за успех Scrum в проекте. Как правило, эту роль в проекте играет менеджер проекта или лидер команды (Team Leader). Важно подчеркнуть, что Scrum Мастер не раздает задачи членам команды. В Scrum команда является самоорганизующейся и самоуправляемой.

Основные обязанности Scrum Мастера таковы:

- создает атмосферу доверия,
- участвует в митингах в качестве фасилитатора - человека, обеспечивающий успешную групповую коммуникацию
- устраняет препятствия
- делает проблемы и открытые вопросы видимыми
- отвечает за соблюдение практик и процесса в команде

Scrum Мастер отслеживает прогресс команды при помощи Sprint Backlog, отмечая статус всех задач в спринте. Scrum Мастер может также помогать заказчику создавать список задач для команды

**Product Owner** - это человек, отвечающий за разработку продукта. Как правило представитель заказчика для заказной разработки. Владелец продукта - это единая точка принятия окончательных решений для команды в проекте, именно поэтому это всегда один человек, а не группа или комитет.

**Команда (Team)** - в методологии Scrum команда является самоорганизующейся и самоуправляемой. Команда берет на себя обязательства по выполнению объема работ на спринт перед Владельцем продукта. Работа команды оценивается как работа единой группы. В Scrum вклад отдельных членов проектной команды не оценивается, так как это разваливает самоорганизацию команды.

Размер команды ограничивается размером группы людей, способных эффективно взаимодействовать лицом к лицу. Типичные

размер команды - 7 плюс минус 2. Команда в Scrum кроссфункциональна. В нее входят люди с различными навыками - разработчики, аналитики, тестировщики. Нет заранее определенных и поделенных ролей в команде, ограничивающих область действий членов команды. Команда состоит из инженеров, которые вносят свой вклад в общий успех проекта в соответствии со своими способностями и проектной необходимостью.

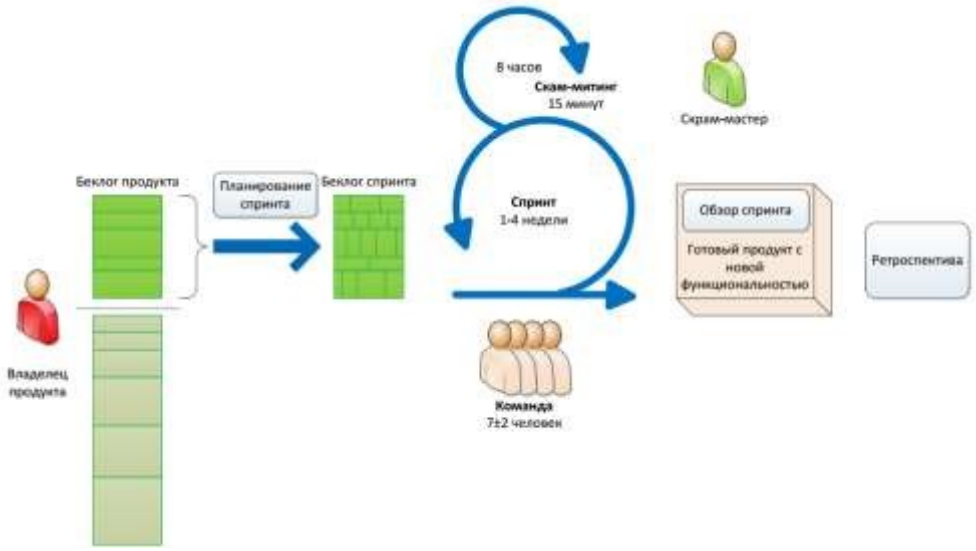


Рис. 3.6 Scrum

В основе лежат короткие ежедневные встречи – Scrum и циклические 30-дневные встречи, называемые спринтом. Результатом спринта является готовый продукт, который можно передавать заказчику (по крайней мере, система должна быть готова к показу заказчику) [25, 26].

Короткие спринты обеспечивают быструю обратную связь проектной команды с заказчиком. Заказчик получает возможность гибко управлять системой, оценивая результат спринта и предлагая улучшения к созданной функциональности. Такие улучшения попадают в список имеющихся на данный момент бизнес-требований и технических требований к системе (Product Backlog), приоритезируются наравне с прочими требованиями и могут быть запланированы на следующий (или на один из следующих) спринтов. Каждый спринт представляет собой маленький «водопад». В течение спринта делаются все работы по сбору требований, дизайну, кодированию и тестированию продукта.

Цель спринта должна быть фиксированным. Это позволяет команде давать обязательства на тот объем работ, который должен быть сделан в

спринте. Это означает, что Sprint Backlog не может быть изменен никем, кроме команды.

### 3.2.4. Экстремальное программирование (eXtreme Programming)

Методология XP, разработанная Кентом Беком (Kent Beck), Уордом Каннингемом (Ward Cunningham) и Роном Джеффрисом (Ron Jeffries), является сегодня одной из самых популярных гибких методологий. Она описывается как набор практик: игра в планирование, короткие релизы, метафоры, простой дизайн, переработки кода (refactoring), разработка «тестами вперед», парное программирование, коллективное владение кодом, 40-часовая рабочая неделя, постоянное присутствие заказчика и стандарты кода.

Интерес к XP рос снизу вверх – от разработчиков и тестировщиков, замученных тягостным процессом, документацией, метриками и прочим формализмом. Они не отрицали дисциплину, но не желали бессмысленно соблюдать формальные требования и искали новые быстрые и гибкие подходы к разработке высококачественных программ.

При использовании XP тщательное предварительное проектирование ПО заменяется, с одной стороны, постоянным присутствием в команде заказчика, готового ответить на любой вопрос и оценить любой прототип, а с другой – регулярными переработками кода (так называемый рефакторинг). Основой проектной документации считается тщательно прокомментированный код. Очень большое внимание в методологии уделяется тестированию. Как правило, для каждого нового метода сначала пишется тест, а потом уже разрабатывается собственно код метода до тех пор, пока тест не начнет выполняться успешно. Эти тесты сохраняются в наборах, которые автоматически выполняются после любого изменения кода.

Хотя парное программирование и 40-часовая рабочая неделя и являются, возможно, наиболее известными чертами XP, но все же носят вспомогательный характер и способствуют высокой производительности разработчиков и сокращению количества ошибок при разработке [26].

### 3.2.5. Crystal Clear

Легковесная гибкая методология, созданная Алистером Коуберном, которая предназначена для небольших команд в 6-8 человек для разработки некритичных бизнес-приложений. Как и все гибкие методологии, Crystal Clear больше опирается на людей, чем на процессы

и артефакты. Crystal Clear использует семь методов/практик, три из которых являются обязательными:

- **частая поставка продукта;**
- **улучшения через рефлексию;**
- **личные коммуникации;**
- чувство безопасности;
- фокусировка;
- простой доступ к экспертам;
- качественное техническое окружение.

Методология Crystal Clear уступает XP по производительности, зато максимально проста в использовании. Она требует минимальных усилий для внедрения, поскольку ориентирована на человеческие привычки. Считается, что эта методология описывает тот естественный порядок разработки ПО, который устанавливается в достаточно квалифицированных коллективах, если в них не занимаются целенаправленным внедрением другой методологии.

Основные характеристики Crystal Clear:

- итеративная инкрементная разработка;
- автоматическое регрессионное тестирование;
- пользователи привлекаются к активному участию в проекте;
- состав документации определяется участниками проекта;
- как правило, используются средства контроля версий кода.

В графическом виде практики Crystal Clear можно изобразить таким образом: