УНИВЕРСИТЕТ ИТМО

И. А. ПЕРЛ
О. В. КАЛЁНОВА

ВВЕДЕНИЕ В МЕТОДОЛОГИЮ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Санкт-Петербург

2019

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

И. А. ПЕРЛ
О. В. КАЛЁНОВА

ВВЕДЕНИЕ В МЕТОДОЛОГИЮ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Учебное пособие

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлениям подготовки 09.03.04 «Программная инженерия»
и 09.04.04 «Программная инженерия» в качестве учебного пособия
для реализации основных профессиональных образовательных программ
высшего образования бакалавриата

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург

2019

Перл И. А., Калёнова О. В. Введение в методологию программной инженерии: Учебное пособие. – СПб. : Университет ИТМО, 2019. – 53 с.

Рецензент: Пенской А. В., к.т.н., доцент факультета программной инженерии и компьютерного моделирования университета ИТМО.

В учебном пособии рассмотрены основные классические модели разработки программного обеспечения, с их достоинствами, недостатками и рекомендациями к применению в разных ситуациях.

УНИВЕРСИТЕТ ИТМО

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

Содержание

Введение

В индустрии разработки программного обеспечения часто можно встретить термин «методология разработки программного обеспечения» (Software Development Methodology, SDM). Методология разработки программного обеспечения – это структура, используемая для организации, планирования и управления процессом разработки информационной системы. Независимо от того, выбираете вы водопадную модель (Waterfall), итеративную, гибкую или какую-либо другую, насколько хорошо вы придерживаетесь SDM, – вы можете эффективно определить успех или неудачу проекта и/или компании.

В наши дни количество проектов и/или компаний, которые не могут последовательно придерживаться процесса, очень велико. Инициативы по разработке программного обеспечения разрабатываются и осуществляются с использованием подхода «сделать за ночь». Это, как правило, связано с тем, что клиенты не могут в полной мере осознать важность и ценность многочисленных анализов концепций, бизнес-требований, анализа прецедентов и/или совещаний по спецификации дизайна, которые необходимы для производства качественного продукта. Вместо этого, раз за разом, они чувствуют, что могут сказать: «Я хочу....», и разработчики могут понять это и дать им то, что они хотят, без каких-либо вопросов. Разработка продукта без следования четкой методологии разработки часто приводит к тому, что системы поставляются с опозданием, превышают бюджет и, во многих случаях, не отвечают ожиданиям клиентов или конечных пользователей. Это даже может привести к полному провалу проекта.

Соблюдение правильно определенной методологии позволяет членам проекта давать более точные оценки, разрабатывать стабильные системы, информировать клиента, создавать четкое понимание предстоящей задачи и выявлять подводные камни раньше, предоставляя достаточно времени для внесения корректив. По мере роста и созревания методологии компания становится более эффективной в выявлении потенциальных проблем до их возникновения, что улучшает ее способность активно реагировать на них и разрабатывать более эффективные обходные пути, когда проблемы случаются.

Когда компания или проект не следует определенной методологии или не реализует ее должным образом, по мере продолжения развития все более и более распространенными становятся различные проблемы. Например,

отсутствие надлежащей связи между заказчиком и командами разработчиков часто приводит к тому, что системы не отвечают потребностям заказчика. Недоверие со стороны персонала по управлению клиентами может повлиять на подрядчика по разработке, успешно поддерживающего или получающего последующий контракт. Кроме того, отсутствие базовых методологических концепций или процессов, таких как внутренний процесс коллегиального обзора, часто приводит к развертыванию программного обеспечения с многочисленными дефектами. Поставка нестабильной системы плохо сказывается на репутации как компании, так и разработчиков.

Не менее важным является то, что многие разработчики часто страдают от побочных эффектов, таких как невысокий моральный дух или отсутствие мотивации из-за изменения объема или необходимости постоянных переделок из-за плохо определенных или плохо реализованных процессов, несмотря на усилия по их развитию. Использование четкой методологии разработки может значительно уменьшить эту проблему. При наличии правильно определенных процессов возникает четкий план, который позволяет лучше управлять текущими задачами проекта или компании, а также задавать четкий и понятный всем участникам вектор развития. Конечно, в ходе проекта всегда могут возникнуть незапланированные ситуации, но соблюдение процесса, безусловно, минимизирует эти события.

Основа для последующего успеха компании или проекта заключается в том, чтобы начать использовать методологию разработки программного обеспечения. При этом понятия и шаги, определенные в этой методологии, не должны восприниматься как некие формальности, которые надо показывать «для галочки», – методология должна быть принята командой разработчиков и должна описывать то, как на самом деле ведется работа в компании. Важно также понимать, что не существует единой методологии, которая бы подходила всем проектам и всем компаниям. Более того, очень редко какая-либо из формально описанных методологий реализуется в компаниях в чистом виде, то есть точно так, как она описана. Применяя ту или иную методологию, компании и проекты в большинстве случаев создают ее уникальную адаптированную версию, которая позволяет наиболее эффективно решать уникальные задачи этой конкретной компании. Методология разработки программного обеспечения выступает отправной точкой и должна быть адаптирована для удовлетворения конкретных потребностей проекта.

Данное методическое учебное пособие разработано для поддержки оразовательного процесса по направлениям 09.03.04 и 09.04.04 «Программная инженерия» и содержит в себе теоретические материалы необходимые для основения дисциплин в магистратуре и бакалавриате. Перечень дисциплин включает в себя:

1. Бакалавриат:
    a. Основы проектирования киберфизических систем (3 семестр)
    b. Теория информационной безопасности и методология защиты информации (4 семестр)
2. Магистратура
    a. Системная программная инженерия (1 семестр)
    b. Методология программной инженерии (2-3 семестр)

Жизненный цикл разработки программного обеспечения

Жизненный цикл разработки программного обеспечения (Software Development Life Cycle, SDLC) направлен на создание высококачественной системы, которая отвечает ожиданиям клиентов, эффективно и действенно работает в текущей и планируемой инфраструктуре информационных технологий и является недорогой в обслуживании и экономически эффективной.

Полностью определенный в 1971 году, термин возник в 1960-х годах, когда мэйнфреймы заполнили целые комнаты и возникла настоятельная необходимость определить процессы и оборудование, направленные на создание крупных бизнес-систем. В те дни команды были небольшими, централизованными, и пользователи были «менее» требовательными. Такой сценарий означает, что не было реальной необходимости в уточнении методологий для управления жизненным циклом развития системы. Однако технология развивалась, системы становились все сложнее, и пользователи привыкли к хорошо функционирующим технологиям. Были разработаны модели и процессы для управления деятельностью компаний в рамках организованного жизненного цикла разработки программного обеспечения. Сегодня традиционные подходы к разработке технологических систем скорректированы с учетом постоянно меняющихся сложных потребностей каждой уникальной организации и ее пользователей [1, 2].

Жизненный цикл разработки программного обеспечения – это процесс, используемый индустрией программного обеспечения для того, чтобы проектировать, разворачивать, разрабатывать и тестировать высококачественное программное обеспечение.

- SDLC является общепризнанной аббревиатурой жизненного цикла разработки программного обеспечения.
- SDLC также называется процессом разработки программного обеспечения.
- SDLC – это структура, определяющая задачи, которые выполняют на каждом этапе процесса разработки программного обеспечения.
- ISO / IEC 12207 является международным стандартом для процессов жизненного цикла программного обеспечения. Он является стандартом, который определяет все задачи, необходимые для разработки и поддержки программного обеспечения.

Работа SDLC направлена на снижение стоимости разработки программного обеспечения, одновременно улучшая качество и сокращая время разработки. Достижение этих явно расходящихся целей происходит благодаря следованию плана, который устраняет типичные подводные камни для проектов по разработке программного обеспечения. Этот план включает в себя:

1. Оценку существующих аналогичных систем на предмет их недостатков.
2. Определение требования к новой системе.
3. Создание программного обеспечения на этапах проектирования, разработки, тестирования и развертывания.

Так как этот план основан на анализе опыта большого количества компаний и проектов, то он позволяет избежать различных типичных дорогостоящих ошибок, таких, например, как дефекты, которые обнаружены пользователями уже после выхода продукта.

Схема на рисунке 1 показывает основные стадии жизненного цикла разработки программного обеспечения.

Планирование и анализ требований

Анализ требований – наиболее важный и фундаментальный этап жизненного цикла. Он выполняется старшими членами команды с входными данными от клиента, отдела продаж, исследований рынка и экспертов в

целевой отрасли. Эта информация затем используется для планирования базового проектного подхода и проведения технико-экономического обоснования продукта в экономической, эксплуатационной и технической областях.

Планирование требований для обеспечения качества и выявление рисков, связанных с проектом, также осуществляется на этом этапе. Результатом технико-экономического обоснования является определение разных технических подходов, которые могут быть использованы для успешной реализации проекта с минимальными рисками.

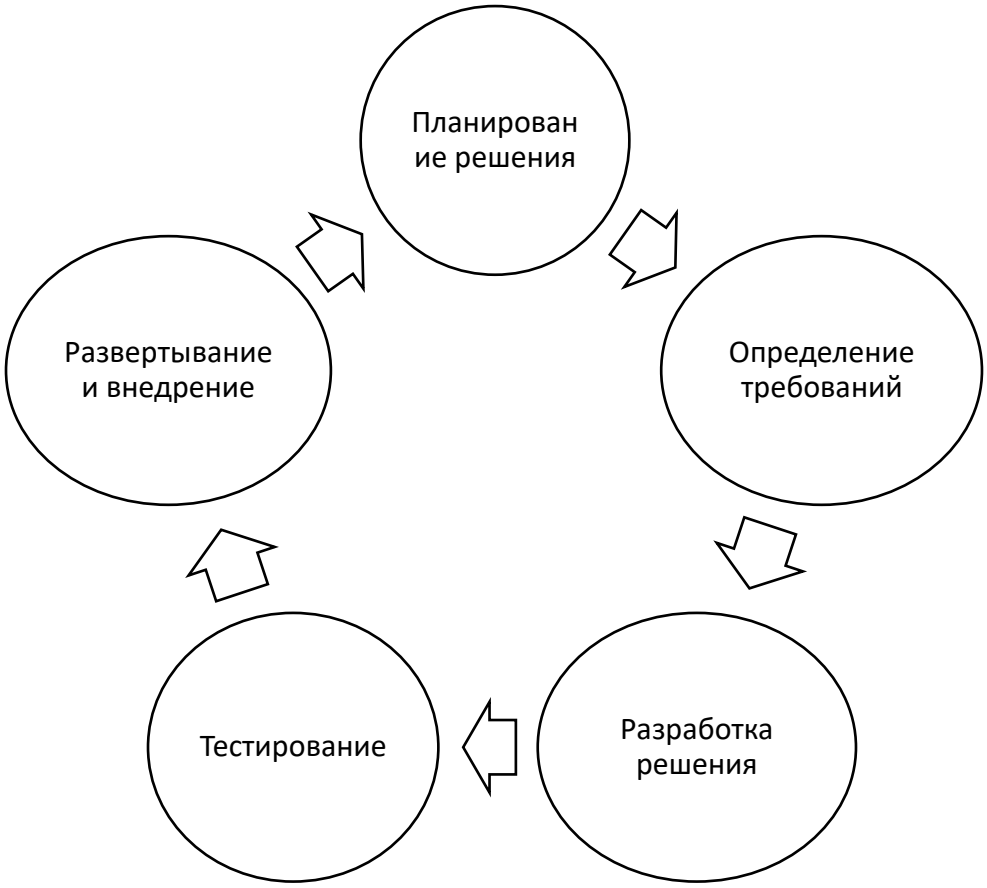

Рисунок 1. Стадии жизненного цикла разработки программного обеспечения

Определенные требования

После проведения анализа требований следующим шагом является четкое определение и документирование требований к продукту и их утверждение заказчиком или аналитиками рынка. Это делается с помощью документа, называемого Спецификация требований к программному продукту (Software Requirement Specification, SRS), который состоит из всех

требований к продукту, которые должны быть разработаны в течение жизненного цикла проекта.

Проектирование архитектуры продукта

Спецификация требований SRS – это справочная информация для архитекторов, разрабатывающих архитектуру продукта. На основе требований, указанных в Спецификации, архитекторами обычно предлагается и документируется более одного подхода к проектированию архитектуры продукта в спецификации разрабатываемого решения (Design Document Specification, DDS).

Спецификация разрабатываемого решения рассматривается всеми важными заинтересованными сторонами, такими как заказчики проекта, субподрядчики, разработчики, специалисты по качеству и внедрению будущего продукта, и на основе разных параметров, таких как оценка риска, надежность продукта, модульность проектирования, бюджетные и временные ограничения – для продукта выбирается наиболее оптимальный подход к проектированию.

Подход к проектированию четко определяет все архитектурные модули продукта, а также их связь и представление потоков данных с внешними и сторонними модулями (если таковые имеются). Внутренний дизайн всех модулей предлагаемой архитектуры должен быть четко определен с мельчайшими деталями в DDS.

Создание или разработка продукта

На этом этапе SDLC начинается фактическая разработка и создается продукт. Программный код создается в соответствии с DDS. Если конструкция выполнена в детальном и организованном формате, то разработку кода можно выполнить без дополнительных затрат ресурсов и времени на разные согласования.

Разработчики должны следовать рекомендациям по кодированию, определенным их организацией и инструментами программирования, такими как компиляторы, интерпретаторы, отладчики и т. д., используемым для написания кода. Для кодирования используются различные языки программирования высокого уровня, такие как C, C++, C#, Java, PHP и пр. Язык программирования выбирается в зависимости от типа разрабатываемого программного обеспечения и требований к нему.

Тестирование продукта

Этот этап обычно является подмножеством всех этапов; как и в современных моделях SDLC, тестовые мероприятия в основном участвуют во всех этапах SDLC. Однако этот этап относится только к этапу тестирования продукта, на котором дефекты продукта озвучиваются, отслеживаются, исправляются и повторно тестируются до тех пор, пока продукт не достигнет стандартов качества, определенных в спецификации требований.

Развертывание и обслуживание

Как только продукт протестирован и готов к развертыванию, он официально выпускается на соответствующем рынке. Иногда развертывание продукта происходит поэтапно в соответствии с бизнес-стратегией организации. Сначала продукт может быть выпущен в ограниченном сегменте и протестирован в реальной бизнес-среде (приемочное тестирование – User Acceptance Testing, UAT).

Затем, основываясь на обратной связи, продукт может быть выпущен «как есть» или с предлагаемыми улучшениями на целевой сегмент рынка. После того как продукт выпущен на рынок, его жизненный цикл переходит на стадию поддержки. Поддержка программного продукта включает в себя не только устранение дефектов, найденных пользователями, или добавление улучшений, предложенных ими, но также и добавление новой функциональности, обеспечивающей развитие продукта.

Модели реализации жизненного цикла

Существует ряд моделей, реализующих жизненный цикл разработки программного обеспечения и дающих конкретные рекомендации о том, как именно реализовывать тот или иной этап жизненного цикла. Эти модели также называются моделями процессов разработки программного обеспечения. Каждая модель процесса состоит из серии шагов, уникальных для своего типа, чтобы обеспечить успех в процессе разработки программного обеспечения.

Ниже приведены наиболее известные и популярные модели SDLC, используемые в отрасли:

- водопадная модель (Waterfall),
- итерационная модель (Iterative model),
- спиральная модель (Spiral model),

- V-модель (V-model),
- модель большого взрыва (Big-bang model).

Водопадная модель

Классическая водопадная модель является базовой моделью жизненного цикла разработки программного обеспечения. Это очень простая, но идеалистичная модель. В середине 1990-х она была очень популярна, но в настоящее время используется мало. Однако она остается важной при изучении методологии разработки программного обеспечения, потому что все остальные модели жизненного цикла разработки программного обеспечения основаны в той или иной степени на классической модели водопада.

Классическая модель водопада делит жизненный цикл на множество фаз. Эта модель считает, что один этап может быть запущен только после завершения предыдущего этапа. То есть выход одной фазы будет входом для следующей фазы. Таким образом, процесс развития можно рассматривать как последовательный поток в водопаде. Здесь фазы не перекрываются друг с другом. Разные последовательные фазы классической водопадной модели показаны на рисунке 2.

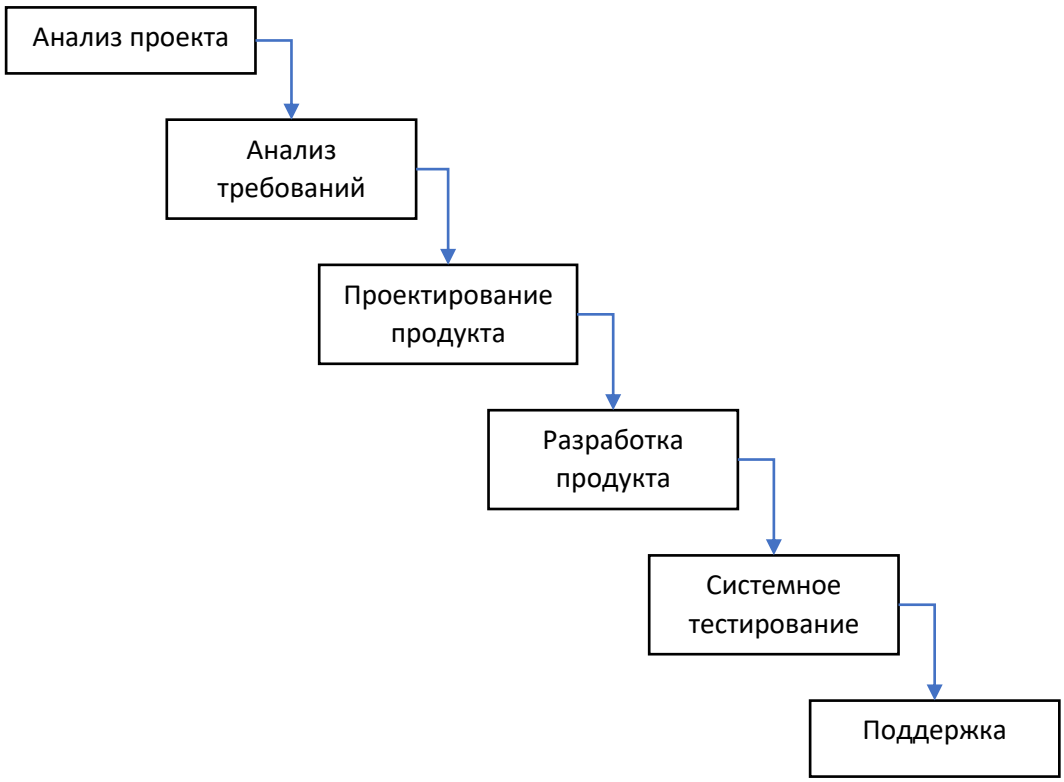

Рисунок 2. Фазы классической водопадной модели

1. Анализ проекта. Основная цель этого этапа заключается в определении того, будет ли разработка программного обеспечения финансово и технически осуществима.
   Технико-экономическое обоснование предполагает понимание проблемы, а затем определение различных возможных стратегий решения проблемы. Эти различные идентифицированные решения анализируются на основе их преимуществ и недостатков, выбирается наиболее оптимальное решение, а все остальные этапы выполняются в соответствии с выбранной стратегией реализации решения.
2. Анализ требований. Целью этапа анализа и спецификации требований является понимание точных требований заказчика и их правильное документирование. Этот этап состоит из двух разных видов деятельности:
   1) Сбор и анализ требований: все требования, касающиеся программного обеспечения, собираются от клиента, а затем анализируются. Целью анализа является устранить неполноту (неполные требование – это требования, в которых некоторые части были опущены) и несоответствия (несогласованность

требований означает, что одна часть требования противоречит другой части).

2) Спецификация требований: проанализированные требования собираются в едином документе, который называется спецификацией требований к программному обеспечению (SRS). Документ SRS служит контрактом между командой разработчиков и клиентами. Любой будущий спор между заказчиками и разработчиками может быть разрешен путем изучения документа SRS.

3. Проектирование продукта. Целью этапа проектирования является преобразование требований, указанных в документе SRS, в структуру, пригодную для реализации на некотором языке программирования.
4. Разработка продукта. На этапе разработки программное обеспечение переводится в исходный код с помощью любого подходящего языка программирования. Таким образом реализуется каждый описанный в спецификации модуль разрабатываемой системы. Также на этом этапе проводится активное модульное тестирование каждой разработанной компоненты. Цель модульного тестирования – проверить, работает ли каждый модуль правильно (в соответствии со спецификацией) или нет.
5. Интеграционное и системное тестирование. Интеграция разных модулей осуществляется вскоре после их кодирования и модульного тестирования. Интеграция разных модулей осуществляется поэтапно. На каждом этапе интеграции в частично интегрированную систему добавляются ранее запланированные модули и тестируется результирующая система. Наконец, после того как все модули успешно интегрированы и протестированы, получена полная рабочая система и проведено тестирование системы.

   Тестирование системы состоит из трех видов тестирования:

   1) α-тестирование – тестирование системы, выполняемое командой разработчиков;
   2) β-испытание: испытание системы, выполненное дружественным коллективом клиентов или пользователей;
   3) приемочное тестирование: после доставки программного обеспечения клиент проводит приемочное тестирование, чтобы определить – принять поставленное программное обеспечение или отклонить его.
6. Поддержка. Техническое обслуживание является наиболее важным этапом жизненного цикла программного обеспечения. Усилия,

затрачиваемые на обслуживание, составляют 60% от общего объема усилий, затрачиваемых на разработку полного программного обеспечения. Существует три основных типа поддержки:

1) Корректирующее обслуживание: этот тип обслуживания выполняется для исправления ошибок, которые были обнаружены не на этапе разработки продукта, а были получены в отчетах от пользователей системы.
2) Развивающая поддержка: данный вид технического обслуживания осуществляется для расширения функциональных возможностей системы на основе запроса клиента.
3) Адаптивное обслуживание: обычно требуется для переноса программного обеспечения для работы в новой среде, такой как работа на новой компьютерной платформе или с новой операционной системой.

Достоинства классической водопадной модели

Классическая водопадная модель – это идеалистическая модель разработки программного обеспечения. Эта модель очень проста, поэтому ее можно рассматривать как основу для других моделей жизненного цикла разработки программного обеспечения. Ниже приведены некоторые основные преимущества этой модели:

- Модель очень проста и понятна.
- Фазы в модели исполняются последовательно по одной.
- Каждый этап модели четко определен.
- Модель имеет очень четкие и понятные вехи.
- Процессы, действия и результаты очень хорошо документированы.
- Укрепляет хорошие привычки: «определись, потом делай» (define before design).
- Модель хорошо работает для небольших проектов и тех проектов, где требования определены и вероятность их изменения мала.

Недостатки классической водопадной модели

Классическая водопадная модель имеет ряд недостатков, не позволяющих использовать ее в чистом виде в реальных проектах. Ниже приведены некоторые основные недостатки модели:

- Нет обратной связи: в классической модели водопада эволюция программного обеспечения от одной фазы к другой похожа на водопад. Предполагается, что разработчики никогда не совершают ошибок на каких-либо этапах. Таким образом, отсутствует какой-либо механизм исправления ошибок.
- Трудно удовлетворить запросы на изменение: эта модель предполагает, что все требования клиентов могут быть полностью и правильно определены в начале проекта, но на самом деле требования клиентов продолжают меняться со временем. Трудно удовлетворить любые запросы на изменение после завершения этапа спецификации требований.
- Отсутствие перекрытия между фазами: в соответствии с этой моделью новая фаза может начаться только после завершения предыдущей. Но в реальных проектах это невозможно. Для повышения эффективности и снижения затрат фазам необходимо частично перекрываться.

Итеративная водопадная модель

В практическом проекте разработки программного обеспечения классическая водопадная модель плохо применима. Таким образом, итеративную модель водопада можно рассматривать как включающую необходимые изменения в классическую водопадную модель для ее использования в практических проектах разработки программного обеспечения. Эта модель почти повторяет классическую, с добавлением некоторых изменений для повышения эффективности разработки программного обеспечения.

Итеративная водопадная модель обеспечивает обратные связи от каждой фазы к предыдущим фазам, что является основным отличием от классической модели водопада.

Пути обратной связи, введенные в итеративную модель, показаны на рисунке 3.

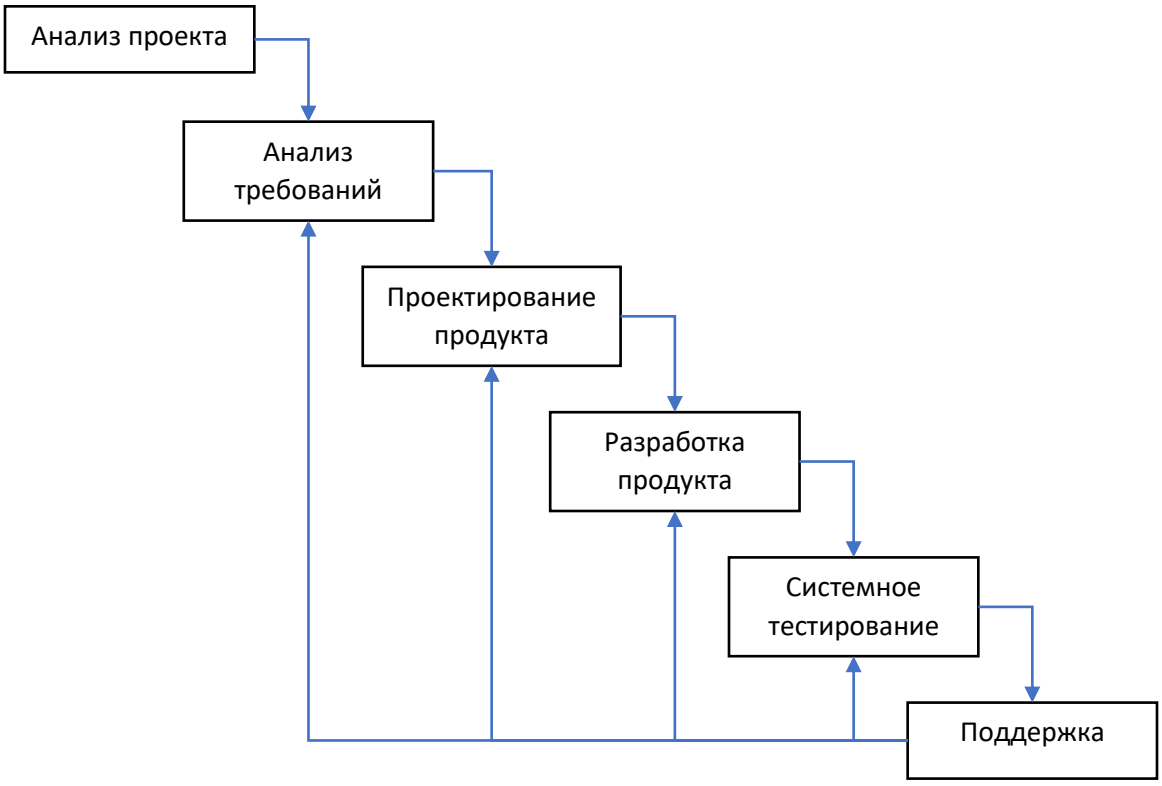

Рисунок 3. Итеративная водопадная модель

Когда ошибки обнаруживаются на более позднем этапе, пути обратной связи позволяют исправить ошибки, допущенные программистами на определенном этапе. Пути обратной связи позволяют переработать фазу, в которой совершены ошибки, и эти изменения отразятся в более поздних фазах. Несмотря на это, обратной связи с этапом технико-экономического обоснования нет, потому что после того как проект принят, отменить его уже нельзя.

Несмотря на наличие обратных связей, необходимо стараться выявлять все ошибки на той стадии, на которой они допущены. Это уменьшает усилия и время, необходимые для их исправления.

Принцип обнаружения ошибок как можно ближе к точкам их возникновения известен как фазовое сдерживание ошибок (в рамках одной фазы).

Достоинства итеративной водопадной модели

- Наличие путей обратной связи: в классической водопадной модели нет путей обратной связи, поэтому нет механизма исправления ошибок. В итеративной модели путь обратной связи от одной фазы к предыдущей позволяет исправить ошибки, и эти изменения отражаются в более поздних фазах.
- Простота: итеративная модель водопада очень проста в понимании и использовании. Вот почему это одна из наиболее широко используемых моделей разработки программного обеспечения.

Недостатки итеративной водопадной модели

- Сложность обработки запросов на изменение: основным недостатком итеративной водопадной модели является то, что все требования должны быть четко сформулированы до начала этапа разработки. Клиент может изменить требования через некоторое время, но итеративная модель водопада не оставляет возможностей для включения запросов на изменение, которые выполняются после начала этапа разработки.
- Инкрементная доставка не поддерживается: в итеративной водопадной модели программное обеспечение полностью разработано и протестировано перед доставкой клиенту. Нет возможности для какой-либо промежуточной поставки. Таким образом, клиенты должны долго ждать получения программного обеспечения.
- Перекрытие фаз не поддерживается: итеративная водопадная модель предполагает, что одна фаза может начаться после завершения предыдущей фазы, но в реальных проектах фазы могут перекрываться, чтобы уменьшить усилия и время, необходимые для завершения проекта.
- Обработка рисков не поддерживается: проекты могут страдать от различных видов рисков. Однако итеративная модель водопада не имеет механизма обработки рисков.
- Ограниченное взаимодействие с клиентами: взаимодействие с клиентами происходит в начале проекта во время сбора требований и при завершении проекта во время доставки программного обеспечения. Такое малое количество взаимодействий с клиентами

может привести ко многим проблемам, поскольку окончательно разработанное программное обеспечение может отличаться от реальных требований клиентов.

Спиральная модель

Спиральная модель менее известна по сравнению с некоторыми другими моделями жизненного цикла разработки программного обеспечения, например, такой как водопадная модель. Причиной этого может являться то, что спиральная модель может быть довольно дорогостоящей в использовании и хуже работать для небольших проектов.

Это риск-ориентированная модель, что означает: общий успех проекта в значительной степени зависит от этапа анализа рисков. Анализ рисков требует специальных знаний на каждой итерации. Таким образом, для рассмотрения и анализа проекта периодически требуются специальные навыки.

На первый взгляд может показаться, что эта модель сложная и неуклюжая, и нет причин рассматривать ее как один из вариантов использования в реальном проекте. Но, как любые другие модели SDLC, эта, помимо недостатков, имеет и уникальные сильные стороны.

Например, есть возможность добавить некоторые дополнительные функции на последних этапах разработки программного продукта. Поскольку мониторинг рисков и регулярная экспертиза являются основными характеристиками этого подхода, то общая работа проекта становится более прозрачной.

Спиральная модель может характеризоваться многократным повторением набора элементарных процессов развития и устранением риска, поэтому на протяжении жизненного цикла проекта риски активно сокращаются.

Чтобы понять, как работает спиральная модель, рассмотрим диаграмму на рисунке 4.

Спиральная модель состоит из четырех основных этапов жизненного цикла разработки программного обеспечения. Весь процесс развития

неоднократно проходит через эти стадии. Каждая итерация называется спиралью.

Четыре основных этапа:

1) определение целей,
2) оценка альтернатив,
3) разработка и верификация,
4) планирование.

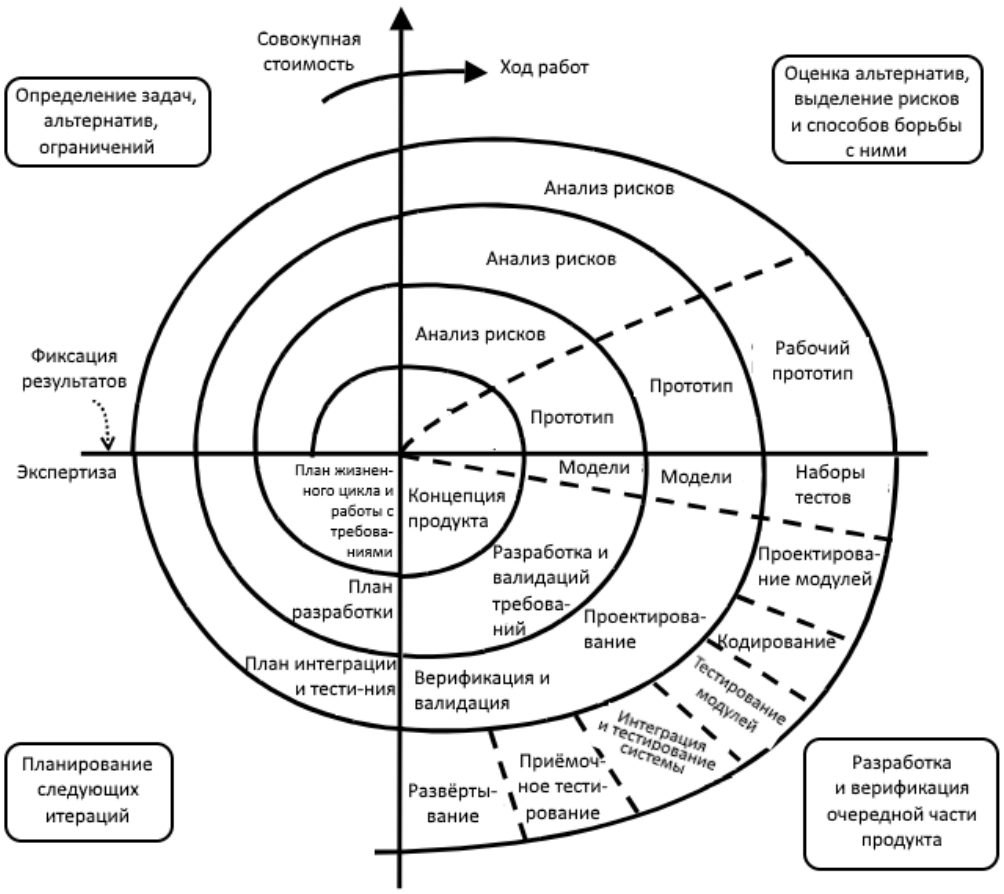

Рисунок 4. Структура спиральной модели

Определение целей

Члены команды пытаются собрать цели продукта, требования (например, спецификации бизнес-требований, спецификации системных требований), альтернативы в дизайне и т. д. В последующих спиралях все требования уточняются согласно обратной связи с клиентом. Таким образом,

постоянное общение между заказчиком и руководством проекта имеет решающее значение.

Оценка альтернатив, выявление, устранение рисков или этап анализа рисков

Данная фаза, вероятно, является наиболее значимым этапом развития. Каковы риски в этом контексте? Риски – это возможные условия и события, которые мешают команде разработчиков достичь своих целей. Их множество – от тривиальных до фатальных. Основная задача группы разработчиков – перечислить все возможные риски и определить их приоритетность в соответствии с дороговизной их реализации. Следующим шагом является определение потенциальных стратегий, которые могут помочь преодолеть риски. Оценка этих параметров может вызвать изменения на следующих этапах. В конце этого этапа производится прототип.

Разработка, проверка продукта следующей итерации или инженерная фаза

На этой стадии производится разработка продукта параллельно с тестированием результата работы предыдущей итерации. Во время первой спирали, когда общие требования не так ясны, разрабатывается так называемое доказательство концепции (Proof of Concept, PoC), для того чтобы получить обратную связь от клиента. Позже, в последующих спиралях, рабочая версия продукта под названием сборка может быть разработана и отправлена клиенту, чтобы получить новую, более подробную обратную связь. Такой подход позволяет добиться большей ясности в требованиях.

Планирование следующего этапа, или этап оценки

Эта фаза позволяет оценить выход проекта «на сегодняшний день», прежде чем проект перейдет к следующей спирали.

Спиральная модель называется метамоделью, потому что она использует как модели водопада, так и прототипы. Но очень важно понимать, что спиральная модель – не просто последовательность приращений водопада. Нисколько.

На самом деле, эта модель довольно гибкая. Вы должны помнить, что диаграмма, о которой мы говорили ранее, содержит некоторые упрощения. Может показаться, что все в проекте следует одной спиральной последовательности, но это не так. Жизненный цикл реального проекта более гибок, чем это простое представление. Есть даже возможность пересмотреть предыдущее решение.

Достоинства спиральной модели

- Мониторинг рисков – одна из основных составляющих, которая делает эту модель довольно привлекательной, особенно когда вы управляете большими и дорогими проектами. Более того, такой подход делает ваш проект прозрачнее, потому что по дизайну каждая спираль должна быть рассмотрена и проанализирована.
- Клиент может видеть рабочий продукт на ранних этапах жизненного цикла разработки программного обеспечения.
- Различные изменения могут быть добавлены на поздних этапах жизненного цикла.
- Проект может быть разделен на несколько частей, и более рискованные из них могут быть разработаны раньше, что уменьшает трудности управления проектом.
- Проектные оценки с точки зрения графика, затрат становятся реалистичнее по мере продвижения проекта, а петли в спирали завершаются.
- Строгий контроль документации.

Недостатки спиральной модели

- Поскольку мониторинг рисков требует дополнительных ресурсов, эта модель может быть довольно дорогостоящей. Каждая спираль требует специальных знаний, что делает процесс управления сложным. Вот почему эта модель SDLC не подходит для небольших проектов.
- Большое количество промежуточных этапов. В результате – огромное количество документации.
- Управление временем может быть трудным. Как правило, дата окончания проекта неизвестна на первых этапах.

Еще один важный момент, о котором следует помнить, – спиральная модель должна использоваться в проектах, для которых она изначально была разработана. Эта модель может быть подходящим вариантом, если вы столкнулись с проектом среднего или высокого риска, затраты же очень важны, клиент не уверен в своих потребностях, а требования сложны и ожидаются значительные изменения.

V-модель

V-модель – это уникальная методология линейной разработки, используемая в течение жизненного цикла разработки программного обеспечения. V-модель фокусируется на довольно типичном методе водопада, который следует строгим пошаговым этапам. Хотя начальные этапы являются широкими этапами проектирования, прогресс продолжается вниз через все более и более детализированные этапы, ведущие к внедрению и кодированию, и, наконец, обратно через все этапы тестирования до завершения проекта.

Подобно традиционной модели водопада, V-модель определяет ряд линейных этапов, которые должны происходить в течение всего жизненного цикла, по одному, пока проект не будет завершен. По этой причине V-модель не считается гибким методом разработки, и из-за огромного объема этапов и их интеграции понимание модели в деталях может быть сложным для всех в команде, не говоря уже о клиентах или пользователях.

Общая структура V-модели показана на диаграмме на рисунке 5.

V-образная форма метода V-модели представляет разные этапы, которые будут пройдены в течение жизненного цикла разработки программного обеспечения. Начиная с верхней левой ступени и двигаясь со временем в направлении верхней правой вершины, ступени представляют собой линейную прогрессию развития, аналогичную водопадной модели.

Рассмотрим каждый из девяти этапов, участвующих в типичной V-модели, и то, как все они объединяются для создания готового продукта.

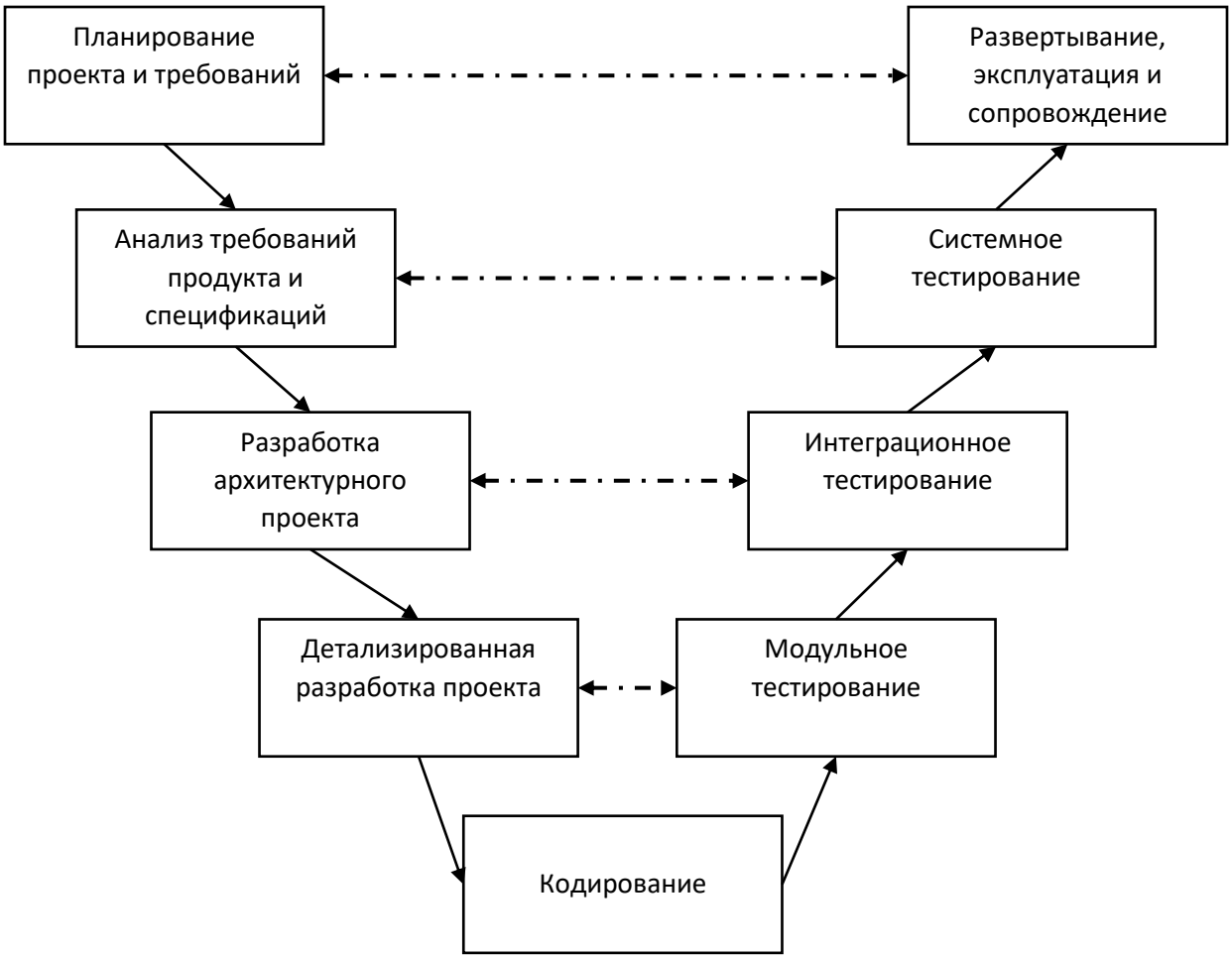

Рисунок 5. Структура V-модели

Планирование проекта и требований

На этом начальном этапе выполняются сбор и анализ системных требований для определения набора функций и потребностей пользователей. Как и в случае с аналогичной фазой из водопадной модели или других подобных методов, следует посвятить много времени тщательному документированию требований пользователя, что на этом этапе имеет решающее значение и делается только один раз.

Еще одним уникальным компонентом V-модели является то, что на каждом этапе проектирования системы разрабатывается спецификация для соответствующих тестов, также предназначенных для последующего внедрения на этапах тестирования. Таким образом, на этапе требований

проектируются приемочные испытания, на этапе разработки спецификации описывается системное тестирование и так далее.

Анализ требований продукта и спецификаций

Используя обратную связь с заказчиком и требования пользователей, собранные, оформленные и согласованные при планировании проектов и требований, этот этап используется для создания документа спецификации, который будет описывать все технические компоненты, такие как слои данных, бизнес-логику и так далее.

На этом этапе также разрабатываются системные тесты для последующего использования.

Разработка архитектурного проекта

На этом этапе составляются спецификации, которые подробно описывают, как в приложении будут взаимодействовать все его компоненты – либо внутри, либо через внешние интеграции. Часто это называют дизайном высокого уровня.

В это же время разрабатываются интеграционные тесты.

Детализированная разработка проекта

Целью этой фазы является низкоуровневое проектирование системы, включая подробные спецификации того, как будет реализована вся функциональная, закодированная бизнес-логика, такая как модели, компоненты, интерфейсы и так далее.

Модульные тесты также должны быть созданы на этапе проектирования модуля.

Кодирование

На этом этапе, на полпути через этапы процесса, происходит фактическая разработка целевой системы. На этот период должно отводиться столько времени, сколько необходимо для преобразования всех ранее созданных проектных и технических документов в кодированную функциональную систему. Этот этап должен быть полностью завершен до начала этапов тестирования.

Модульное тестирование

Теперь процесс движется вверх по дальней стороне V-модели с обратным тестированием, начиная с модульных тестов, разработанных на этапе проектирования модуля. В идеале эта фаза должна устранить подавляющее большинство вероятных ошибок и проблем и, таким образом, стать самой продолжительной фазой тестирования проекта.

Тем не менее, как и при выполнении модульного тестирования с другими моделями разработки, модульные тесты не могут (или не должны) охватывать все возможные проблемы, которые могут возникнуть в системе, поэтому менее детализированные этапы тестирования должны заполнить эти пробелы.

Интеграционное тестирование

Тестирование, разработанное на этапе проектирования архитектуры, выполняется здесь, что гарантирует: система функционирует во всех компонентах и вместе со сторонними интеграциями.

Системное тестирование

Производится тестирование системы в целом с использованием тестов, разработанных на этапе проектирования системы. На этой стадии тестирование в основном фокусируется на производительности и регрессионном тестировании.

Развертывание, эксплуатация и сопровождение

На этой стадии производится финальное тестирование системы. Так называемое приемочное тестирование необходимо, чтобы убедиться, что вся система функционирует в соответствии с требованиями пользователя и заказчик подтверждает, что разработанное решение – это именно то, что ему нужно.

Приемо-сдаточное тестирование – это процесс внедрения всех тестов, созданных на начальном этапе проекта. Оно должно обеспечивать функционирование системы в живой среде с фактическими данными, готовыми к развертыванию.

После этого производится развертывание системы в целевом окружении и проект переходит в состояние поддержки.

Достоинства V-модели

- Подходит для ограниченных проектов: из-за строгой природы V-модели и ее линейного проектирования, реализации и этапов тестирования неудивительно, что V-модель была в значительной степени принята промышленностью медицинских устройств в последние годы. В ситуациях, когда сроки и объем проекта четко определены, технология стабильна, а документация и проектные спецификации ясны, V-модель может быть отличным методом.
- Идеально подходит для управления временем: в том же ключе V-модель также хорошо подходит для проектов, которые должны выдерживать строгие сроки и соответствовать ключевым контрольным датам на протяжении всего процесса, с довольно четкими и хорошо понятными этапами, которые вся команда может легко понять и подготовиться. Менеджерам и инженерам относительно просто создать временную линию для всего жизненного цикла разработки, генерируя контрольные точки для каждого этапа на этом пути. Конечно, использование V-модели никоим образом не гарантирует, что контрольные точки всегда будут выполнены, но строгий характер самой модели заставляет соблюдать довольно плотный график.

Недостатки V-модели

- Нехватка адаптивности: подобно проблемам, стоящим перед традиционной водопадной моделью, на которой основана V-модель, наиболее проблемным аспектом V-модели является ее неспособность адаптироваться к любым необходимым изменениям в течение жизненного цикла разработки. Например, упущенная из виду проблема в рамках какой-либо фундаментальной системы, которая затем обнаруживается только на этапе внедрения, может представлять собой серьезную проблему с точки зрения потерянных человеко-часов, а также возросших затрат.
- Ограничения временной шкалы: хотя это не является неотъемлемой проблемой самой V-модели, но акцент на тестировании в конце

жизненного цикла означает, что слишком легко оказаться привязанным в конце проекта к выполнению тестов в состоянии спешки, чтобы соответствовать назначенным срокам или контрольным точкам.

- Плохо подходит для длительных жизненных циклов: как и водопадная модель, V-модель полностью линейна, и поэтому проекты нельзя легко изменить после того, как поезд разработки покинул станцию отправления. Поэтому V-модель плохо подходит для обработки долгосрочных проектов, которые могут потребовать много версий или постоянных обновлений/исправлений.
- Хотя V-модель, безусловно, не единственная модель разработки, которая подпадает под эту критику, нельзя отрицать, что строгий и методичный характер V-модели и ее разных линейных этапов, как правило, подчеркивают цикл разработки, подходящий менеджерам и пользователям, а не разработчикам и архитекторам. С помощью такого метода, как V-модель, руководителям проектов или другим исполнителям можно легко упустить из виду огромные сложности разработки программного обеспечения в пользу попыток уложиться в сроки или просто чувствовать себя чрезмерно уверенным в ходе работ или текущем прогрессе, основанном исключительно на том, какой этап жизненного цикла активно разрабатывается.

Rational Unified Process (RUP)

Rational Unified Process – это процесс разработки программного обеспечения. Он обеспечивает упорядоченный подход к распределению задач и обязанностей в развитии организации. Его цель заключается в обеспечении производства высококачественного программного обеспечения, отвечающего потребностям конечных пользователей, в рамках предсказуемого графика и бюджета [3].

Rational Unified Process – это продукт процесса, разработанный и поддерживаемый программным обеспечением Rational. Команда разработчиков Rational Unified Process тесно сотрудничает с клиентами, партнерами, группами продуктов Rational, а также с консультантской организацией Rational, чтобы обеспечивать постоянное обновление и

совершенствование процесса с учетом свежего опыта и лучших развивающихся и проверенных практик.

Шесть ключевых практик

В Rational Unified Process описывается, как эффективно применять коммерчески проверенные подходы к разработке программного обеспечения для групп разработчиков программного обеспечения. Их называют «лучшими практиками» не потому, что вы можете точно определить их ценность, а потому, что они обычно используются в промышленности успешными организациями. Rational Unified Process предоставляет каждому члену команды рекомендации, шаблоны и инструменты, необходимые для всей команды, чтобы в полной мере использовать следующие практики:

1. Разрабатывать программное обеспечение итеративно.
2. Управлять требованиями.
3. Использовать модульную архитектуру.
4. Визуально моделировать программное обеспечение.
5. Проверять качество программного обеспечения.
6. Управлять изменениями в программном обеспечении.

Итеративная разработка программного обеспечения

Учитывая современные сложные программные системы, невозможно последовательно сначала определить всю проблему, спроектировать все решение, построить программное обеспечение, а затем в конце протестировать продукт. Требуется итерационный подход, который позволяет повысить понимание проблемы путем последовательных уточнений и постепенного увеличения эффективного решения в течение нескольких итераций. Rational Unified Process поддерживает итеративный подход к разработке, который учитывает элементы наивысшего риска на каждом этапе жизненного цикла и значительно уменьшает степень риска проекта. Этот итеративный подход помогает адресовать риск с помощью наглядного прогресса за счет частых выпусков сборок проекта, которые обеспечивают непрерывное вовлечение конечных пользователей и обратную связь. Поскольку каждая итерация заканчивается выпуском сборки, команда разработчиков сосредотачивается на получении результатов, а частые проверки состояния помогают гарантировать, что проект остается в графике.

Итеративный подход также облегчает учет изменений в требованиях, функциях решения или графике проекта.

Управление требованиями

Rational Unified Process описывает, как выявлять, организовывать и документировать необходимые функциональные возможности и ограничения; отслеживать и документировать компромиссы и решения; и легко фиксировать и сообщать бизнес-требования. Понятия прецедентов и сценариев, описанных в этом процессе, оказались отличным способом учесть функциональные требования и обеспечить выполнение разработки, внедрения и тестирования программного обеспечения, что увеличит вероятность того, что конечная система будет удовлетворять потребности конечных пользователей. Они обеспечивают последовательные и отслеживаемые потоки как через разработку, так и через поставленную пользователю систему.

Модульная архитектура

Процесс фокусируется на ранней разработке и базовой разработке надежной исполняемой архитектуры до выделения ресурсов для полномасштабной разработки. В нем описывается, как разработать гибкую архитектуру, способную приспосабливаться к изменениям, интуитивно понятную и способствующую более эффективному повторному использованию программного обеспечения. Rational Unified Process поддерживает разработку программного обеспечения на основе компонентов. Компоненты – это нетривиальные модули, подсистемы, выполняющие четкую функцию. Процесс Rational Unified обеспечивает системный подход к определению архитектуры с использованием новых и существующих компонентов. Они собраны в четко определенной архитектуре либо в компонентной инфраструктуре, такой как интернет, CORBA и COM, для которой появляется индустрия многократно используемых компонентов.

Визуальное моделирование программного обеспечения

Процесс показывает, как визуально моделировать программное обеспечение для передачи структуры и поведения архитектуры в целом и отдельных компонентов. Это позволяет скрыть детали и написать код,

используя «графические строительные блоки». Визуальные абстракции помогают вам общаться с разными аспектами вашего программного обеспечения; видеть, как элементы системы подходят друг к другу; убедиться, что строительные блоки согласуются с вашим кодом; поддерживать согласованность между дизайном и его реализацией; способствовать однозначной связи. Основой успешного визуального моделирования является отраслевой стандарт Unified Modeling Language (UML), созданный Rational Software.

Проверка качества программного обеспечения

Низкая производительность приложений и низкая надежность являются общими факторами, которые резко тормозят распространение многих современных программных приложений. Следовательно, качество должно быть пересмотрено с учетом требований, основанных на надежности, функциональности, производительности приложений и производительности системы. Rational Unified Process помогает в планировании, проектировании, реализации, выполнении и оценке этих типов тестов. Оценка качества встроена в процесс, во все виды деятельности, с участием всех участников, с использованием объективных измерений и критериев, а не рассматривается как запоздалая мысль или отдельная деятельность, выполняемая отдельной группой.

Управление изменениями в программном обеспечении

Способность управлять изменениями – это уверенность в том, что каждое изменение приемлемо, а способность отслеживать изменения необходима в среде, в которой изменения неизбежны. Процесс описывает, как контролировать и отслеживать изменения, чтобы обеспечить успешную итеративную разработку. Он также поможет вам в создании безопасных рабочих пространств для каждого разработчика, обеспечивая изоляцию от изменений, внесенных в другие рабочие пространства, и контролируя изменения всех программных артефактов (например, моделей, кода, документов и т. д.). И это объединяет команду, помогая участникам работать как единое целое, описывая автоматизацию интеграции и управление сборкой.

Работа процесса в двух измерениях

Процесс может быть описан в двух измерениях, или вдоль двух осей:

- горизонтальная ось представляет время и показывает динамический аспект процесса по мере его осуществления, и он выражается в терминах циклов, фаз, итераций и этапов;
- вертикальная ось представляет статический аспект процесса: он описывается в терминах действий, артефактов и рабочих процессов.

Общая схема двух измерения показана на рисунке 6.

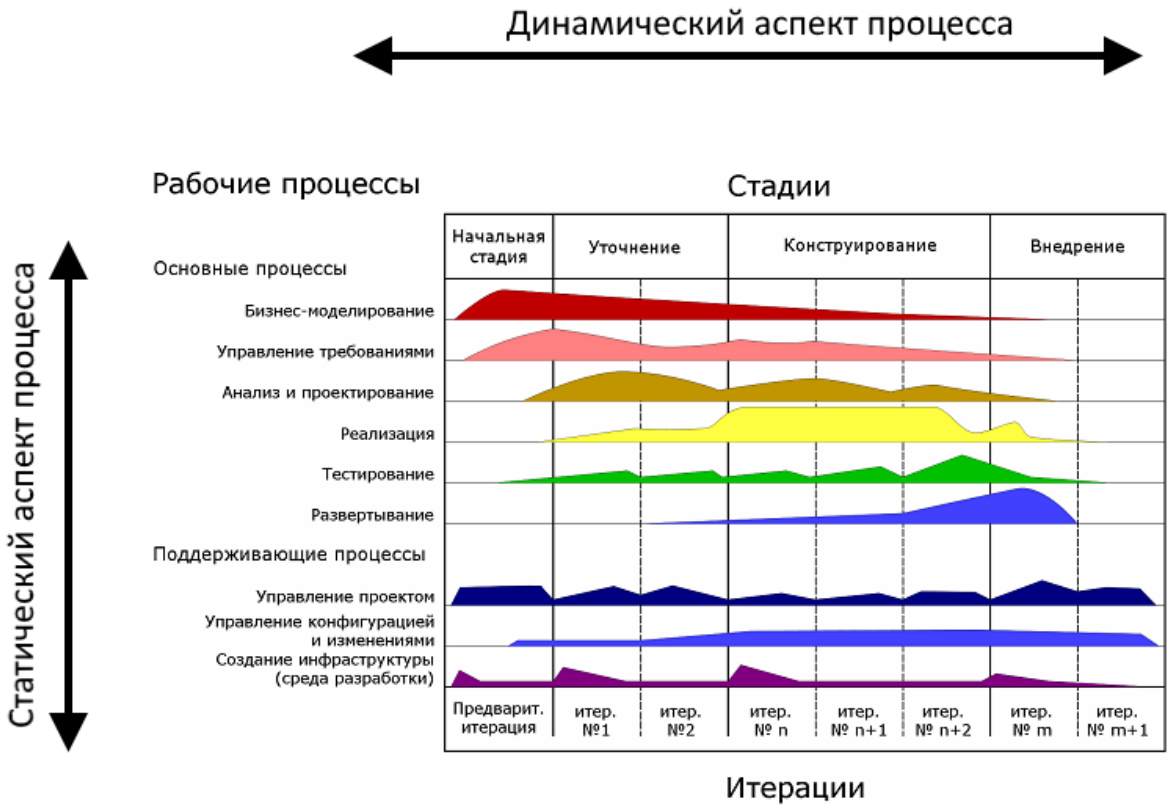

Рисунок 6. Структура процесса RUP в двух измерениях

Процессы и итерации – динамический аспект

Жизненный цикл программного обеспечения разбивается на циклы, каждый из которых работает над новым поколением продукта. Rational

Unified Process делит один цикл разработки на четыре последовательных этапа:

1) начальная фаза,
2) уточнение,
3) конструирование,
4) внедрение.

Каждый этап завершается четко определенной контрольной точкой, когда должны быть приняты определенные важные решения и, следовательно, должны быть достигнуты ключевые цели. На рисунке 7 показана общая организация фаз проекта.

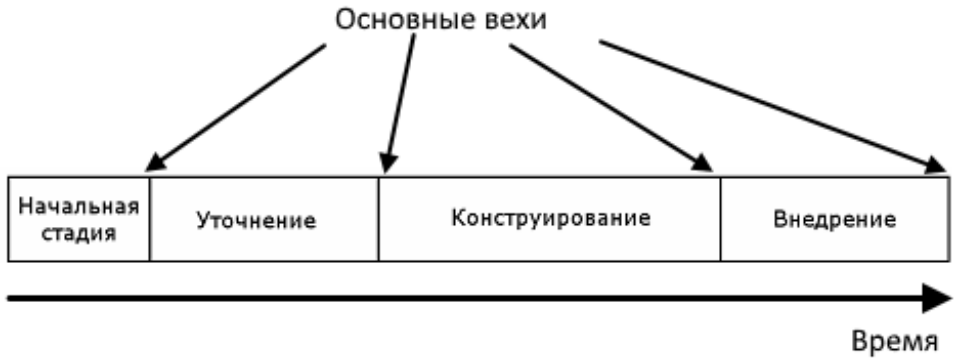

Рисунок 7. Фазы и основные вехи процесса

Начальная стадия

На начальном этапе необходимо выбрать и зафиксировать бизнес-кейс для системы и разграничить область проекта. Для этого необходимо идентифицировать все внешние сущности, с которыми будет взаимодействовать система (акторы), и определить характер этого взаимодействия на высоком уровне. Это включает в себя идентификацию всех вариантов использования и описание нескольких значимых из них. Бизнес-кейс включает критерии успеха, оценку рисков и оценку необходимых ресурсов, а также план этапа с указанием сроков основных этапов. Итогом начального этапа являются:

- Концептуальный документ: общее видение требований, ключевых особенностей и основных ограничений основного проекта.
- Начальная модель сценариев использования (завершена на 10–20%).

- Первоначальный глоссарий проекта (может быть частично выражен в виде модели домена).
- Начальный бизнес-кейс, который включает бизнес-контекст, критерии успеха (прогноз доходов, признание рынка и т. д.) и финансовый прогноз.
- Первоначальная оценка рисков.
- План проекта, показывающий этапы и итерации.
- Бизнес-модель, если необходимо.
- Один или несколько прототипов.

В конце начального этапа находится первая крупная веха проекта: веха целей жизненного цикла. Критерии оценки на начальном этапе:

- Согласованное с заинтересованными сторонами определение сферы охвата и смета расходов/графика.
- Понимание требований, о чем свидетельствует согласованность первичных сценариев использования.
- Достоверность оценок затрат/графиков, приоритетов, рисков и процесса развития.
- Глубина и широта архитектурного прототипа, который был разработан.
- Соотношение фактических расходов против запланированных расходов.

Проект может быть отменен или значительно пересмотрен, если он не пройдет этот этап.

Фаза уточнения

Целью этапа уточнения является анализ проблемной области, создание прочной архитектурной основы, разработка плана проекта и устранение наиболее рискованных элементов проекта. Для достижения этих целей вы должны иметь представление о системе «шириной в милю и глубиной в дюйм». Архитектурные решения должны приниматься с пониманием всей системы: ее объема, основных функциональных и нефункциональных требований, таких как требования к производительности.

Можно утверждать, что этап уточнения является наиболее важным из четырех этапов. В конце этого этапа основная «инженерия» считается завершенной, и проект проходит свой самый важный день расчета: необходимо принять решение о том, следует ли приступать к этапам конструирования и внедрения. Для большинства проектов это также

соответствует переходу от мобильной, легкой и гибкой операции с низким риском к дорогостоящей операции с высоким риском со значительной инерцией. Хотя процесс всегда должен учитывать изменения, мероприятия этапа уточнения гарантируют, что архитектура, требования и планы достаточно стабильны, а риски достаточно сглажены, поэтому вы можете предсказуемо определить стоимость и график завершения разработки. Концептуально этот уровень точности соответствовал бы уровню, необходимому для того, чтобы организация взяла на себя обязательства относительно этапа конструирования с фиксированной ценой.

На этапе уточнения прототип исполняемой архитектуры строится в одной или нескольких итерациях в зависимости от объема, размера, риска и новизны проекта. Эти усилия должны, по крайней мере, касаться критических случаев использования, выявленных на начальном этапе, которые, как правило, сопряжены с серьезными техническими рисками проекта. Хотя эволюционный прототип компонента производства всегда является целью, это не исключает разработки одного или нескольких исследовательских, одноразовых прототипов для смягчения конкретных рисков, таких как компромиссы дизайна или требований, технико-экономическое обоснование компонента или демонстрация инвесторам, клиентам и конечным пользователям.

Итогом этапа уточнения является:

- Модель сценариев использования (не менее 80%) – все сценарии и субъекты определены, большинство описаний сценариев разработаны.
- Дополнительные требования, охватывающие нефункциональные требования и любые требования, не связанные с конкретным вариантом использования.
- Описание архитектуры программного обеспечения.
- Исполняемый архитектурный прототип.
- Пересмотренный перечень рисков и пересмотренное деловое обоснование.
- План развития общего проекта, включая план проекта, показывающий итерации и критерии оценки каждой итерации.
- Обновленный случай разработки, определяющий используемый процесс.
- Предварительное руководство пользователя (необязательно).

В конце этапа уточнения находится вторая важная веха проекта – веха архитектуры жизненного цикла. На этом этапе вы изучаете подробные цели и область применения системы, выбор архитектуры и разрешение основных рисков.

Основные критерии оценки на этапе уточнения включают ответы на следующие вопросы:

- Понимание продукта стабилизировано?
- Архитектура стабильна?
- Показывает ли исполняемый прототип, что основные элементы риска рассмотрены и достоверно разрешены?
- Достаточно ли детализирован и точен план этапа конструирования? Подкреплен ли он достоверными оценками?
- Согласны ли все заинтересованные стороны с тем, что нынешнее понимание может быть достигнуто, если будет выполнен нынешний план развития всей системы в контексте нынешней архитектуры?
- Приемлемы ли фактические расходы на ресурсы по сравнению с запланированными расходами?

Проект может быть прерван или значительно переосмыслен, если он не пройдет эту веху.

Фаза конструирования

Во время фазы конструирования все остальные компоненты и функции системы разработаны и интегрированы в продукт, и вся функциональность тщательно протестирована. Этап конструирования – это, в некотором смысле, производственный процесс, где акцент делается на управлении ресурсами и управлении операциями для оптимизации затрат, графиков и качества. В этом смысле основная задача управления претерпевает переход от развития интеллектуальной собственности во время создания и уточнения к разработке внедряемых продуктов во время разработки и внедрения.

Многие проекты достаточно велики, чтобы можно было вести параллельную разработку отдельных компонентов. Эти параллельные действия могут значительно ускорить доступность развертываемых сборок; они также могут повысить сложность управления ресурсами и синхронизации рабочих процессов. Надежная архитектура и понятный план тесно взаимосвязаны. Другими словами, одним из важнейших качеств

архитектуры является простота ее реализации. Это одна из причин, почему сбалансированное развитие архитектуры и плана важны на этапе уточнения. Результатом этапа конструирования является продукт, готовый к передаче конечным пользователям. Как минимум, он состоит:

- Из программного продукта, интегрированного на соответствующих целевых платформах.
- Руководства пользователя.
- Описания текущего выпуска.

В конце этапа конструирования находится третья крупная проектная веха (начальная веха эксплуатации). На этом этапе вы решаете, готовы ли программное обеспечение, окружение и пользователи к работе, не подвергая проект высоким рискам. Этот релиз часто называют «бета».

Критерии оценки этапа конструирования включают ответы на следующие вопросы:

- Является ли этот выпуск продукта стабильным и достаточно зрелым для развертывания в сообществе пользователей?
- Готовы ли все заинтересованные стороны к переходу в сообщество пользователей?
- По-прежнему ли приемлемы фактические расходы ресурсов по сравнению с запланированными расходами?

Внедрение может быть отложено на один релиз, если проект не достиг этой вехи.

Фаза внедрения

Целью фазы внедрения является передача программного продукта пользователям. После передачи продукта конечному пользователю обычно возникают проблемы, требующие разработки новых выпусков, исправления некоторых проблем или завершения отложенных функций.

Внедрение производится тогда, когда продукт является достаточно зрелым для развертывания в домене конечного пользователя. Для этого обычно требуется, чтобы какая-то полезная часть системы была завершена до приемлемого уровня и качества и чтобы была доступна пользовательская документация, с тем чтобы переход к пользователю обеспечил положительные результаты для всех сторон.

Внедрение включает:

- «бета-тестирование» для проверки новой системы на соответствие ожиданиям пользователей;
- параллельная работа с устаревшей системой, которую она заменяет;
- миграция требуемых баз данных;
- обучение пользователей и операторов;
- развертывание продукта для групп маркетинга, дистрибуции и продаж.

На этапе перехода основное внимание уделяется деятельности, необходимой для передачи программного обеспечения в руки пользователей. Как правило, этот этап включает несколько итераций, включая бета-версии, версии общей доступности, а также исправления ошибок и улучшения. Значительные усилия затрачиваются на разработку ориентированной на пользователя документации, обучение пользователей, поддержку пользователей в их первоначальном использовании продукта и реагирование на их отзывы. Однако на данном этапе жизненного цикла отзывы пользователей должны ограничиваться в первую очередь вопросами настройки, установки и удобства использования продукта.

Основные цели этапа внедрения включают:

1. Достижение самостоятельности пользователя.
2. Достижение согласия заинтересованных сторон в том, что исходные условия развертывания являются полными и соответствуют критериям оценки концепции.
3. Достижение базового уровня конечного продукта как можно быстрее и экономичнее.

Эта фаза может варьироваться от очень простой до чрезвычайно сложной, в зависимости от типа продукта. Например, новый выпуск существующего настольного продукта может быть очень простым, в то время как замена национальной системы управления воздушным движением будет очень сложной.

В конце этапа внедрения находится четвертая важная веха проекта – веха выпуска продукта. На этом этапе вы решаете, были ли достигнуты цели или требуется начать другой цикл разработки. В некоторых случаях этот этап может совпадать с завершением начального этапа следующего цикла.

Основные критерии оценки переходного этапа включают ответы на следующие вопросы:

- Пользователь удовлетворен?
- По-прежнему ли приемлемы фактические расходы по сравнению с запланированными расходами?

Итерации

Каждый этап (фазу) Rational Unified Process можно разбить на итерации. Итерация – это полный цикл разработки, приводящий к выпуску (внутреннему или внешнему) исполняемого продукта, подмножества разрабатываемого конечного продукта, который постепенно растет от итерации к итерации, чтобы стать конечной системой.

По сравнению с традиционным водопадным процессом итеративный процесс имеет следующие преимущества:

- раннее смягчение рисков,
- изменения более управляемы,
- более высокий уровень повторного использования разрабатываемого кода,
- команда проекта может учиться в процессе работы,
- лучшее общее качество.

Структура процесса – статический аспект

Процесс описывает, кто что делает, как и когда. Процесс Rational Unified представлен четырьмя основными элементами моделирования:

- исполнитель – кто,
- активность – как,
- артефакт – что,
- рабочий процесс – когда.

Активности, артефакты и исполнители

Исполнитель

Исполнитель определяет поведение и обязанности человека или группы людей, работающих вместе как команда. Вы можете рассматривать исполнителя как «шляпу», которую человек может носить в проекте.

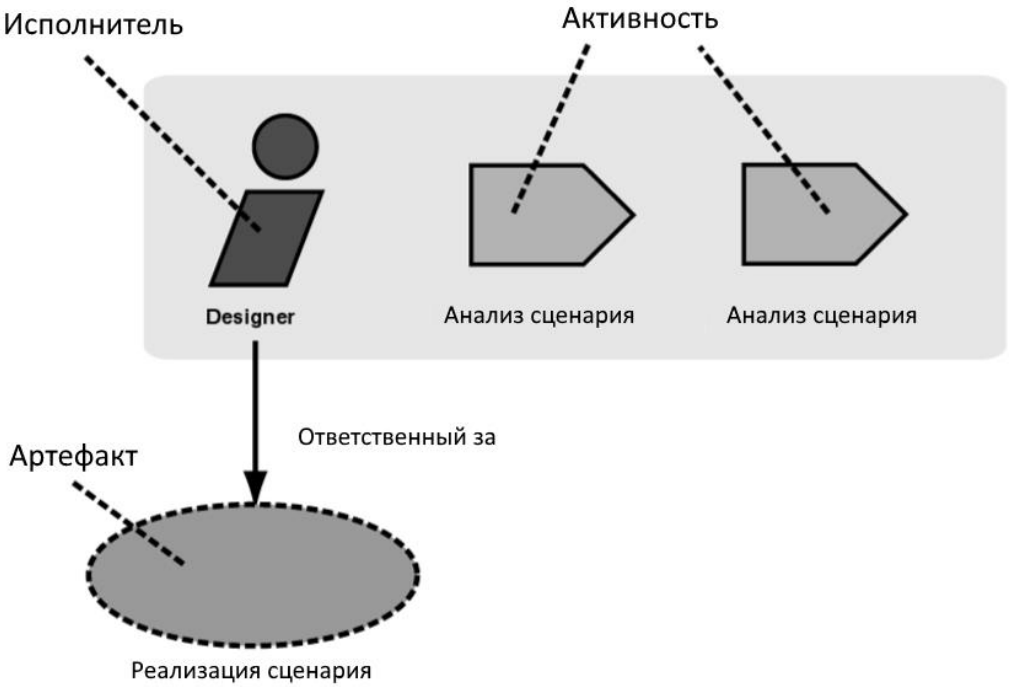

Рисунок 8. Активности, артефакты и исполнители

Один человек может носить много разных «шляп». Это важное различие, потому что естественно думать об исполнителе как о личности или команде, но в Rational Unified Process исполнитель является скорее ролью, определяющей, как люди должны выполнять работу. Обязанности, которые мы возлагаем на исполнителя, включают как выполнение определенного набора действий, так и владение набором артефактов.

Активность

Активность конкретного исполнителя – это единица работы, которая может быть предложена человеку в этой роли. Действие имеет четкую цель, обычно выраженную в создании или обновлении некоторых артефактов, таких как модель, класс, план. Каждое действие назначается конкретному работнику. Общая структура активности показана на рисунке 8. Детализация деятельности обычно составляет от нескольких часов до нескольких дней, она обычно включает одного исполнителя и затрагивает один артефакт. Активность должна использоваться в качестве элемента планирования и

прогресса; если она слишком мала, ею пренебрегают, а если слишком велика, прогресс должен быть выражен в терминах частей активности.

Рисунок 9. Сотрудники и исполнители

Пример активности:

- Планирование итерации для исполнителя: менеджер проекта.
- Поиск вариантов использования и участников для исполнителя: системный аналитик.
- Анализ дизайна, для исполнителя: Design Reviewer.
- Исполнение теста производительности для исполнителя: Тестировщик производительности.

Пример соответствия сотрудников, исполнителей и активностей показан на рисунке 9.

Артефакт

Артефакт – это часть информации, которая создается, изменяется или используется процессом. Артефакты – это материальные продукты проекта,

объекты, которые проект производит или использует при работе над конечным продуктом. Артефакты используются исполнителями в качестве входных данных для выполнения действия и являются результатом таких активностей. В объектно-ориентированных терминах проектирования, поскольку действия являются операциями над активным объектом (исполнителем), артефакты являются параметрами этих активностей.

Рабочие процессы

Простое перечисление всех исполнителей, активностей и артефактов не полностью покроет собой описание процесса. Нам нужен способ описания осмысленных последовательностей активностей, которые дают некоторый ценный результат и показывают взаимодействие между исполнителями.

Рабочий процесс – это последовательность действий, которая создает наблюдаемый результат.

В терминах UML рабочий процесс может быть выражен в виде диаграммы последовательностей, диаграммы совместной работы или диаграммы действий.

Обратите внимание, что не всегда возможно или практично представлять все зависимости между активностями. Часто два вида активностей переплетены теснее, чем показано на рисунке 10, особенно когда в них участвует один и тот же исполнитель. Люди – не машины, рабочий процесс не может быть истолкован буквально как программа для людей, которой нужно следовать точно и механически.

В Rational Unified Process существует девять основных рабочих процессов, которые представляют собой разделение всех работников и действий на логические группы.

Основные рабочие процессы разделены на шесть «инженерных» рабочих процессов:

1) бизнес-моделирование,
2) работы с требованиями,
3) анализ и проектирование,
4) реализация,
5) тестирование,
6) внедрение.

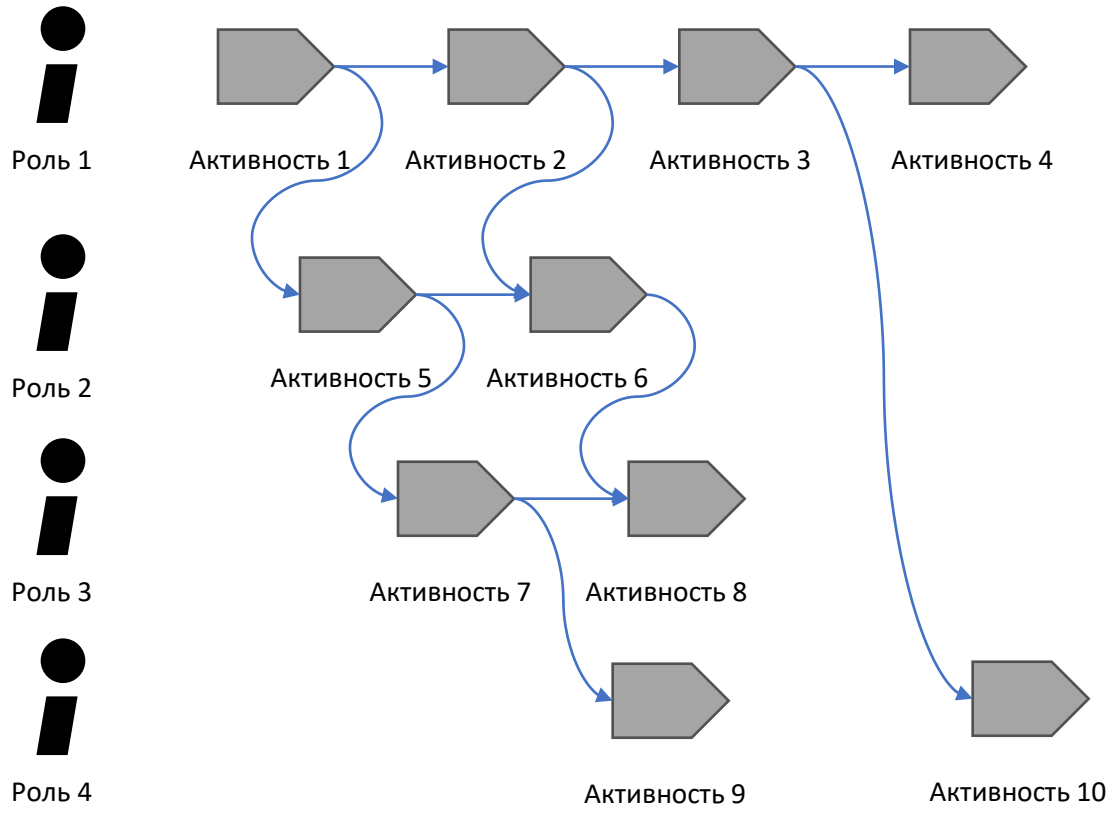

Рисунок 10. Пример взаимосвязи рабочих процессов

И три основных «поддерживающих» рабочих процесса:

1) управление проектами,
2) управление конфигурацией и изменениями,
3) проектное окружение.

Хотя названия шести основных инженерных рабочих процессов могут напоминать последовательные фазы в традиционном водопадном процессе, мы должны иметь в виду, что фазы итеративного процесса отличаются и что эти рабочие процессы пересматриваются снова и снова на протяжении всего жизненного цикла. Фактический полный рабочий процесс проекта переплетает эти девять основных рабочих процессов и повторяет их с разным акцентом и интенсивностью на каждой итерации.

Бизнес-моделирование

Одна из основных проблем большинства инженерных проектов, ориентированных на решение бизнес-задач, заключается в том, что разработчики программного обеспечения и бизнес-инжиниринговое сообщество не общаются должным образом друг с другом. Это приводит к тому, что результат бизнес-инжиниринга не используется должным образом в качестве вклада в усилия по разработке программного обеспечения, и наоборот. Rational Unified Process решает эту проблему, предоставляя общий язык и процесс для обоих сообществ, а также показывая, как создавать и поддерживать прямую связь между бизнес-моделями и моделями программного обеспечения.

В бизнес-моделировании мы документируем бизнес-процессы, используя так называемые бизнес-кейсы. Это обеспечивает общее понимание всеми заинтересованными сторонами того, какой бизнес-процесс необходимо поддерживать в организации. Бизнес-примеры использования анализируются, чтобы понять, как бизнес должен поддерживать бизнес-процессы. Это задокументировано в бизнес-объектной модели. Многие проекты могут отказаться от бизнес-моделирования.

Требования

Цель рабочего процесса требований – описать, что должна делать система, и позволять разработчикам и заказчику согласовать это описание. Для этого мы выявляем, организуем и документируем необходимые функциональные возможности и ограничения; отслеживаем и документируем компромиссы и решения.

Создается концептуальный документ и выявляются потребности заинтересованных сторон. Определяются субъекты, представляющие пользователей и любую другую систему, которая может взаимодействовать с разрабатываемой системой. Определяются варианты использования, представляющие поведение системы. Поскольку варианты использования разрабатываются в соответствии с потребностями субъекта, система, скорее всего, будет актуальна для пользователей. На рисунке 11 показан пример модели сценария для системы машин по переработке.

Каждый вариант использования подробно описан. Описание прецедента показывает, как система взаимодействует шаг за шагом с

действующими лицами и что делает система. Нефункциональные требования описаны в дополнительных спецификациях.

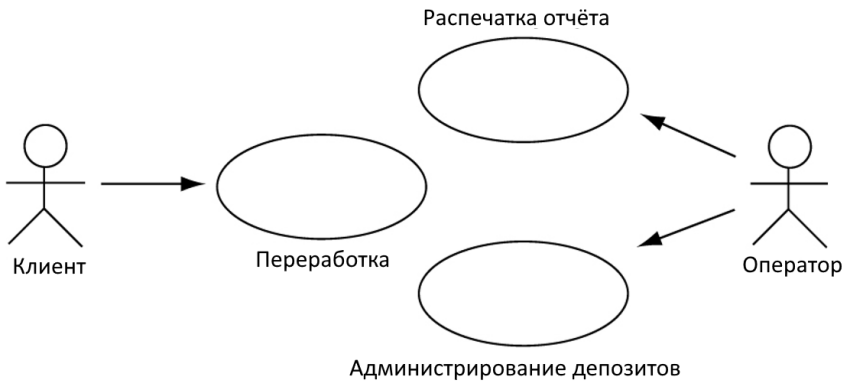

Рисунок 11. Пример модели с участниками и сценариями использования

Варианты использования функционируют как объединяющий поток на протяжении всего цикла разработки системы. Такая же модель прецедента использована во время исследования, анализа и дизайна требований, тестирования.

Анализ и проектирование

Цель рабочего процесса анализа и проектирования – показать, как система будет реализована на этапе реализации. Вы хотите построить систему, которая:

- Выполняет в конкретной среде реализации задачи и функции, указанные в описаниях вариантов использования.
- Выполняет все предъявленные требования.
- Позволяет вносить изменения и улучшения относительно недорого.

Модель проектирования служит абстракцией исходного кода; то есть модель проектирования действует как «схема» того, как исходный код структурирован и написан.

Модель проектирования состоит из классов проектирования, структурированных в пакеты проектирования и подсистемы проектирования с четко определенными интерфейсами, представляющими, что станет с компонентами реализации. Она также содержит описания того, как объекты

этих классов дизайна взаимодействуют для обеспечения вариантов использования.

Проектная деятельность сосредоточена вокруг понятия архитектуры. Производство и проверка этой архитектуры являются основным направлением ранних итераций проектирования. Архитектура представлена рядом архитектурных видов. Эти виды охватывают основные конструктивные решения. По сути, архитектурные перспективы являются абстракциями или упрощениями всего видения, в которых важные характеристики становятся более заметными, оставляя детали в стороне. Архитектура является важным средством не только для разработки хорошей модели дизайна, но и для повышения качества любой модели, построенной во время разработки системы.

Реализация

Основные цели реализации:

- Определить организацию кода с точки зрения реализации подсистем, организованных слоями.
- Реализация классов и объектов с точки зрения компонентов (исходные файлы, двоичные файлы, исполняемые файлы и другие).
- Протестировать разработанные компоненты по отдельности.
- Интегрировать результаты, полученные отдельными исполнителями (или командами), в исполняемую систему.

Система реализуется путем внедрения компонентов. Процесс Rational Unified описывает повторное использование существующих компонентов или внедрение новых компонентов с четко определенной ответственностью, что упрощает обслуживание системы и расширяет возможности повторного использования.

Компоненты структурированы в подсистемы реализации. Подсистемы принимают форму каталогов с дополнительной структурной или управленческой информацией. Например, подсистема может быть создана как каталог или папка в файловой системе, или подсистема в Rational / Apex для C++ или Ada, или пакеты с использованием Java.

Тестирование

Перед тестированием ставятся следующие задачи:

- Проверка взаимодействия между объектами.
- Проверка правильной интеграции всех компонентов программного обеспечения.
- Проверка правильности выполнения всех требований.
- Выявление и обеспечение устранения дефектов до развертывания программного обеспечения.

Процесс Rational Unified предлагает итеративный подход, который означает, что вы тестируете на протяжении всего проекта. Это позволяет находить дефекты как можно раньше, что радикально снижает затраты на их исправление. Тестирование распределяется по трем ключевым направлениям: функциональность, представление применения и представление системы. Для каждого из этих измерений качества процесс описывает, как проходит тестовый жизненный цикл планирования, проектирования, реализации, выполнения и оценки.

В рамках процесса описаны стратегии для того, когда и как автоматизировать тестирование. Автоматизация тестирования особенно важна при использовании итерационного подхода, позволяющего проводить регрессионное тестирование в конце каждой итерации, а также для каждой новой версии продукта.

Внедрение

Целью рабочего процесса внедрения является успешное создание выпусков продуктов и доставка программного обеспечения конечным пользователям. Процесс охватывает широкий круг мероприятий, включая:

- Выпуск внешних выпусков программного обеспечения.
- Упаковку программного обеспечения.
- Распространение программного обеспечения.
- Установку программного обеспечения.
- Оказание помощи и содействия пользователям.

Во многих случаях это также включает такие мероприятия, как:

  - Планирование и проведение бета-тестирования.
  - Миграцию существующего программного обеспечения или данных.
  - Приемку.

Хотя мероприятия по развертыванию в основном сосредоточены на переходном этапе, многие из них необходимо включить в более ранние этапы подготовки к развертыванию в конце этапа строительства. Рабочие процессы развертывания и среды Rational Unified Process содержат меньше деталей, чем другие рабочие процессы.

Управления проектами

Управление проектами программного обеспечения – это искусство балансирования конкурирующих целей, управления рисками и преодоления ограничений для успешной доставки продукта, в котором удовлетворяются потребности как заказчиков, так и пользователей. Тот факт, что так мало проектов бесспорно успешны, достаточно комментирует сложность задач.

Этот рабочий процесс фокусируется главным образом на конкретном аспекте итеративного процесса разработки. Цель этого аспекта – сделать задачу проще, предоставляя:

- основы для управления проектами, требующими больших затрат на программное обеспечение;
- практические рекомендации по планированию, укомплектованию штата, осуществлению и мониторингу проектов;
- системы управления рисками.

Это не рецепт успеха, но он представляет собой подход к управлению проектами, что значительно повышает шансы на успех при разработке программного обеспечения.

Управление конфигурациями и изменениями

В этом рабочем процессе описывается, как управлять многочисленными артефактами, созданными многими людьми, работающими над общим проектом. Контроль помогает избежать дорогостоящей путаницы и гарантирует, что результирующие артефакты не конфликтуют из-за каких-либо проблем, например:

- Одновременное обновление – когда два или более работника работают отдельно над одним и тем же артефактом, и последний вносит изменения, разрушая работу первого.

- Ограниченное уведомление – когда проблема устранена в артефактах, совместно используемых несколькими разработчиками, и некоторые из них не уведомлены об изменении.
- Несколько версий – большинство крупных программ разрабатываются в эволюционных выпусках. Один выпуск может использоваться клиентом, в то время как другой тестируется, а третий еще находится в разработке. Если в одной из версий обнаруживаются проблемы, то между ними необходимо распространять исправления. Путаница может привести к дорогостоящим исправлениям и повторной работе, если изменения не будут тщательно контролироваться.

Этот рабочий процесс содержит рекомендации по управлению несколькими вариантами разрабатываемых программных систем, отслеживанию версий, используемых в данных сборках программного обеспечения, выполнению сборок отдельных программ или целых выпусков в соответствии с пользовательскими спецификациями версий, а также применению политик разработки для конкретного сайта. Рабочие процессы этой категории описывают, как можно управлять параллельной разработкой, разработкой на нескольких сайтах и автоматизировать процесс сборки. Это особенно важно в итерационном процессе, где вы можете захотеть делать сборки часто, что стало бы невозможным без мощной автоматизации.

Также эти процессы включают механизмы, позволяющие вести аудит того, почему, когда и кем был изменен какой-либо артефакт. Этот рабочий процесс также охватывает управление запросами на изменение, т. е. как сообщать о дефектах, управлять ими в течение их жизненного цикла и как использовать данные о дефектах для отслеживания прогресса и тенденций.

Управление окружением

Целью рабочего процесса по управлению окружением является обеспечение организации разработки программного обеспечения окружением разработки программного обеспечения – как процессами, так и инструментами, которые необходимы для поддержки команды разработчиков.

Этот рабочий процесс фокусируется на действиях по настройке процесса в контексте проекта. Он также акцентирует внимание на деятельности по разработке руководящих принципов, необходимых для

поддержки проекта. Процессом Rational Unified Process предоставляется пошаговая процедура, описывающая, как вы реализуете процесс в организации.

Рабочий процесс управления окружением также содержит набор средств разработки, содержащих рекомендации, шаблоны и инструменты, необходимые для настройки процесса.

Некоторые аспекты рабочего процесса управления окружением не охватываются такими процессами, как выбор, приобретение и обеспечение работы инструментов, поддержание среды разработки.

С более подробным описанием структуры процесса Rational Unified Process можно ознакомиться в перечне ключевых стандартов по управлению процессом разработки и качества программных продуктов, приведенном, например в [4].

Вопросы для самоконтроля

1. Перечислите основные этапы классического жизненного цикла проекта и кратко охарактеризуйте их.
2. Классическая водопадная модель, фазы модели, её достоинства и нодостатки.
3. Итеративная водопадная модель, фазы модели. Сравнение с классической водопадной моделью.
4. Дайте определение спиральной модели, опишите 4 основные фазы этой модели. Достоинства и недостаки этой модели.
5. Понятие рисков в рамках процесса разработки программного обеспечения. Работа с рисками в спиральной модели.
6. Общая характеристика V-модели, её этапы и связи между ними. Область применения V-модели.
7. Дайте общую характеристику модели Rational Unified Process, назовите ключевые особенности этой модели в сравнении с моделими, которые предшествовали её появлению (Водопадная модель, Спиральная и V модели).
8. Назовите и охарактеризуйте шесть ключевых практик процесса Rational Unified process.

9. Опишите взаимосвязь между сотрудниками, ролями и активностями. Объясните эти понятия с точки зрения процесса Rational Unified Process.
10. Опишите динамический и статический аспекты модели разработки Rational Unified Process. Перечислите и коротко охарактеризуйте элементы каждого аспекта.

Рекомендуемая литература

1. Шопырин Д. Г. Управление проектами разработки ПО. Дисциплина «Гибкие технологии разработки программного обеспечения» [Электронный ресурс]. – СПб. : НИУ ИТМО, 2007. – 131 с. – Режим доступа: http://e.lanbook.com/book/43554. – Загл. с экрана.
2. Амблер С. Гибкие технологии: экстремальное программирование и унифицированный процесс разработки. – СПб. : Питер, 2005.
3. Крачтен Ф. Введение в Rational Unified Process. – М. : Вильямс, 2002. – 240 с.
4. International standards, approaches and frameworks relevant to Software Quality Management and Software Process Improvement [Электронный ресурс]. – Режим доступа: http://docplayer.net/5026515-International-standards-approaches-and-frameworks-relevant-to-software-quality-management-and-software-process-improvement.html (дата обращения : 10.04.2019).

Иван Андреевич Перл
Ольга Вячеславовна Калёнова

ВВЕДЕНИЕ В МЕТОДОЛОГИЮ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Учебное пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н. Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел

Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49