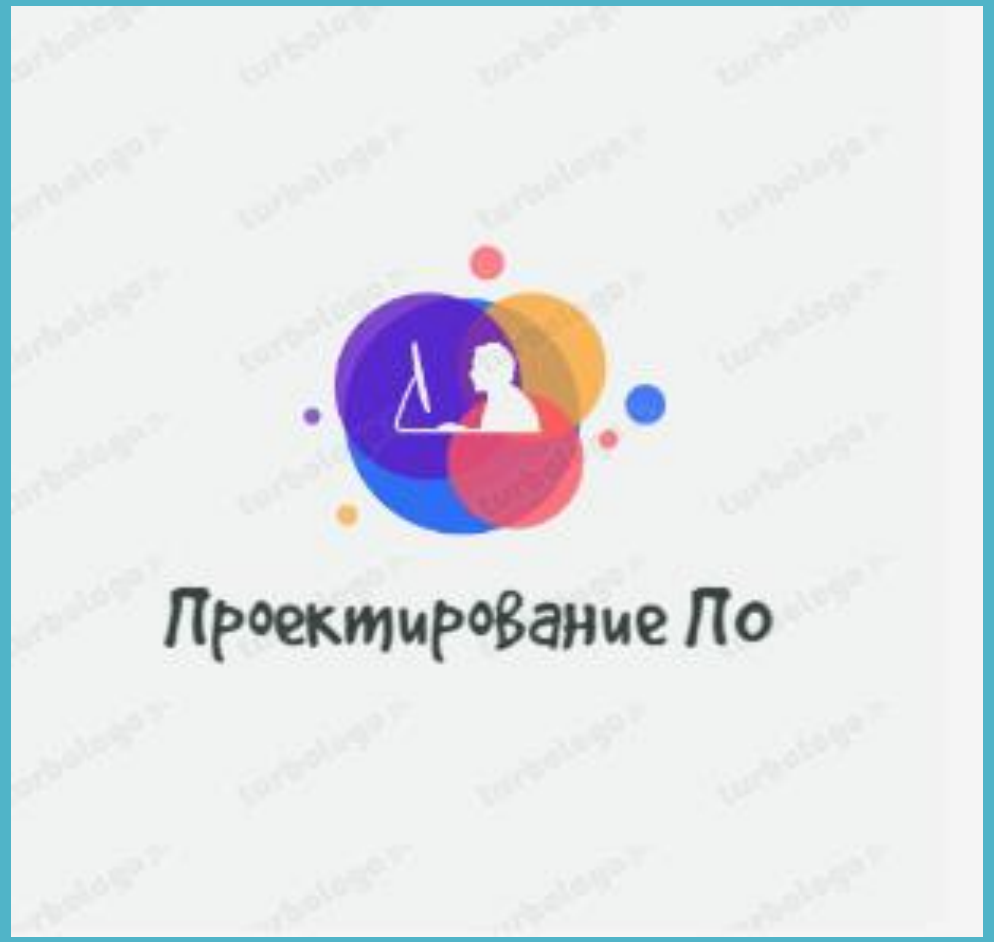
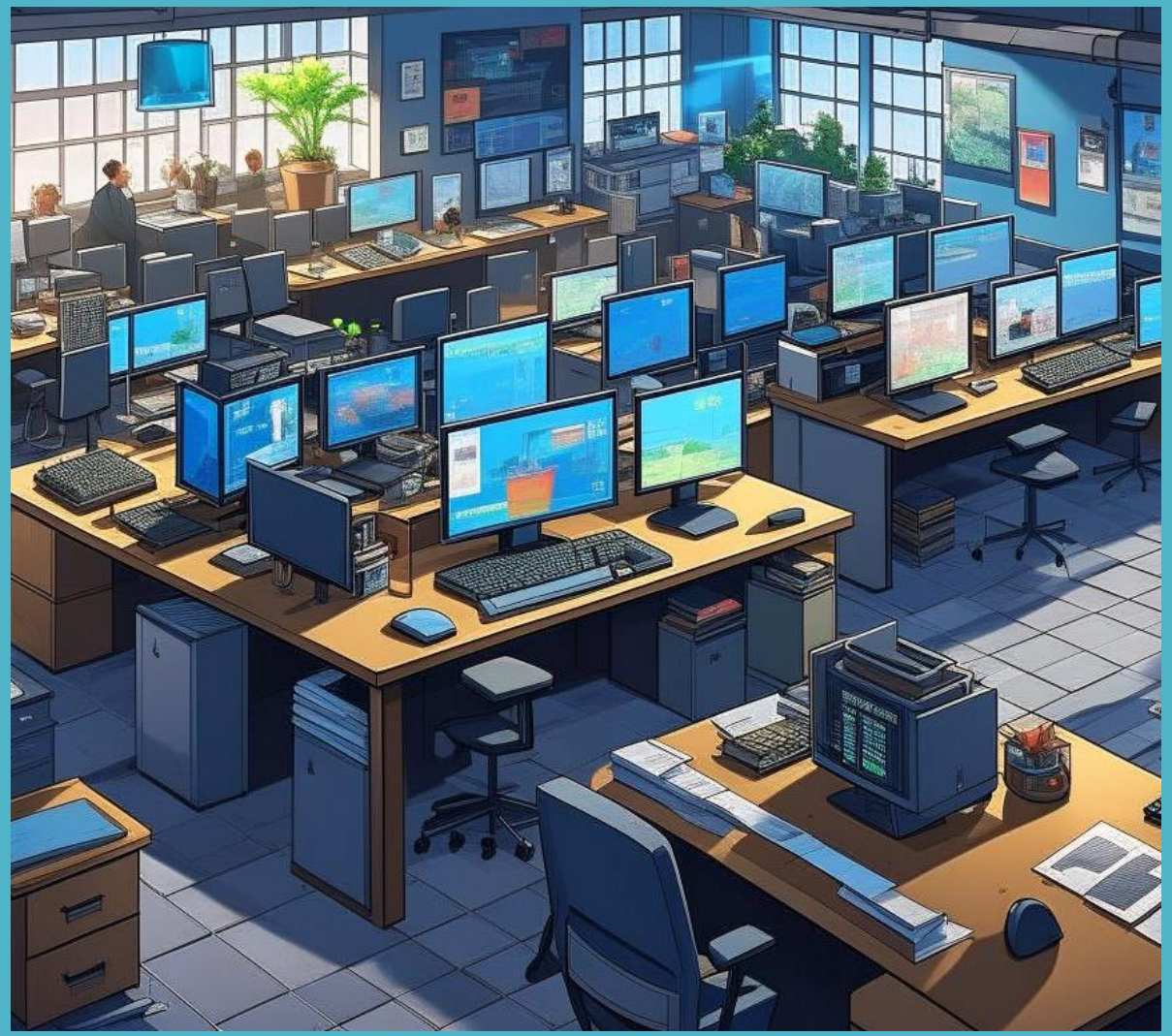
Тема 1: Введение в разработку ПО
Занятие 1.2. Обзор жизненного цикла разработки программного обеспечения.



Цели занятия:

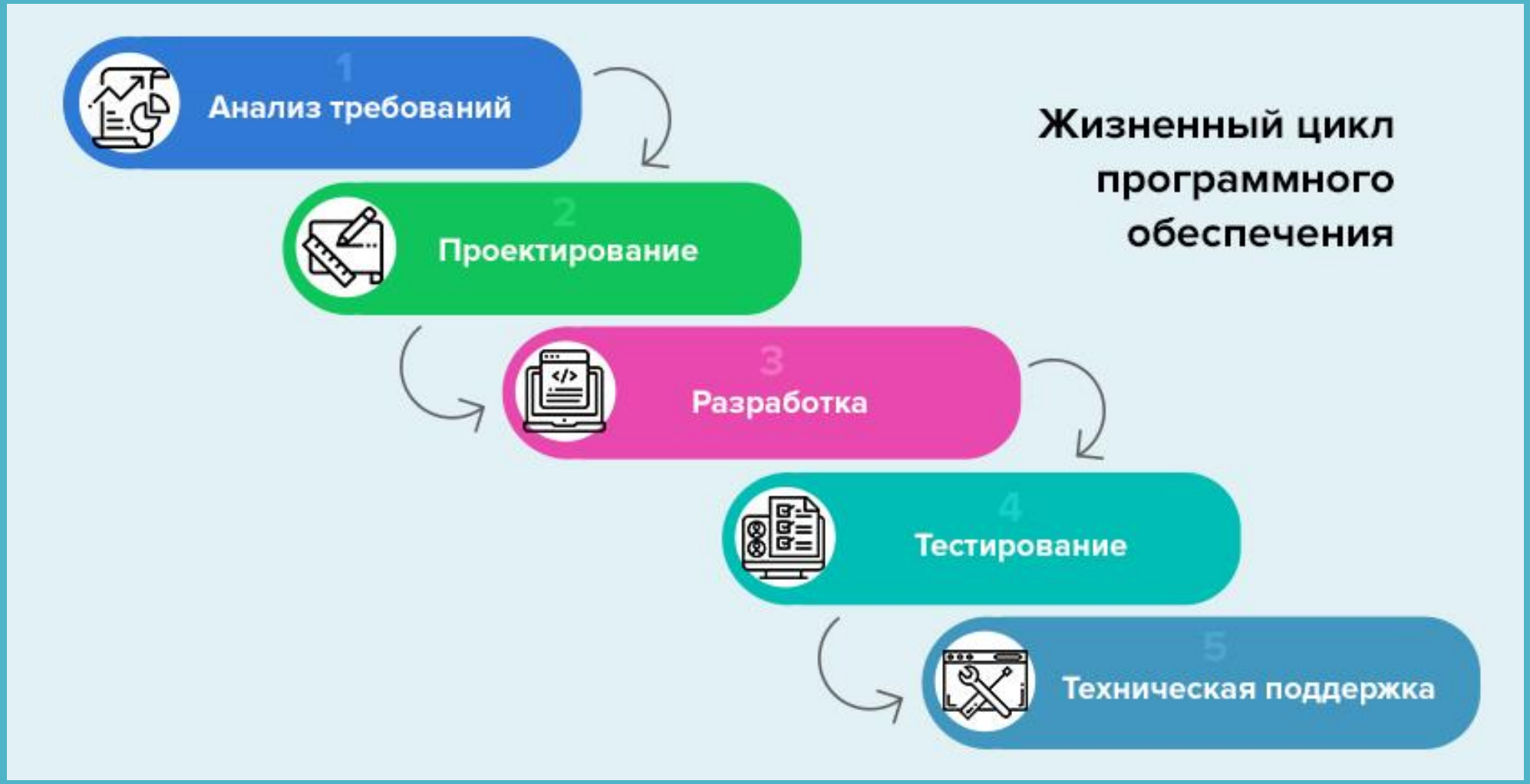
- Познакомиться с основными этапами жизненного цикла разработки программного обеспечения.
- Понять значение каждого этапа и их последовательность.
- Рассмотреть различные модели жизненного цикла разработки ПО.



Учебные вопросы:

1. Понятие жизненного цикла разработки ПО
2. Основные этапы жизненного цикла разработки ПО
3. Модели разработки ПО

1.Понятие жизненного цикла разработки ПО



1.Понятие жизненного цикла разработки ПО

Понятие жизненного цикла разработки программного обеспечения (ПО) отражает последовательность этапов и фаз, через которые проходит программный продукт, начиная с момента его создания и заканчивая завершением его использования.

Этот подход предоставляет организационную и методологическую структуру для планирования, разработки, тестирования, внедрения и поддержки программных проектов.

Понятие жизненного цикла разработки ПО

Жизненный цикл ПО описывает весь спектр деятельности, начиная с идеи создания программного продукта и заканчивая его выводом из эксплуатации. Важно понимать, что этот цикл не ограничивается только техническими аспектами разработки, но также включает в себя управленческие и организационные аспекты.

2.Основные этапы жизненного цикла разработки ПО:

1. Анализ и сбор требований: В начале цикла определяются требования к программному продукту. Этот этап включает в себя определение функциональности, особенностей, характеристик и ожиданий заказчика.

Анализ и сбор требований

Анализ и сбор требований - это один из первых и критически важных этапов в жизненном цикле разработки программного обеспечения. На этом этапе определяются и документируются функциональные и нефункциональные требования к программному продукту, которые служат основой для всего последующего процесса разработки.

Анализ и сбор требований

Функциональные требования определяют, какие функции и возможности должны быть реализованы в программе. Это включает в себя детализированные описания, что продукт должен делать, какие операции и действия он должен поддерживать.

Анализ и сбор требований

Нефункциональные требования определяют характеристики продукта, которые не связаны непосредственно с его функциональностью. Это может включать в себя требования к производительности, безопасности, надежности, масштабируемости, доступности и другие аспекты.

Этапы процесса анализа и сбора требований:

Идентификация заинтересованных: Определение лиц, групп и организаций, чьи потребности, ожидания и требования влияют на продукт.

Сбор информации: Взаимодействие с заинтересованными сторонами для получения информации о том, что должен делать программный продукт, какие задачи должны выполняться и какие цели достигаться.

Документирование требований: Создание документа, в котором подробно описаны все функциональные и нефункциональные требования. Этот документ будет служить основой для разработчиков, тестировщиков и других участников проекта.

Проверка и верификация требований: Убедиться в том, что документ с требованиями полный, понятный и достаточно детализированный. Проверить, что требования реалистичны и выполнимы.

Этапы процесса анализа и сбора требований:

Валидация требований: Удостовериться, что собранные требования действительно соответствуют потребностям заказчика и ожиданиям пользователей. Это может включать в себя демонстрацию или обсуждение с заказчиком.

Управление изменениями требований: По мере развития проекта требования могут изменяться. Важно внимательно следить за изменениями, оценивать их влияние на проект и обеспечивать согласованность между всеми заинтересованными сторонами.

Анализ и сбор требований играют ключевую роль в успешной разработке программного продукта. Ясное понимание потребностей пользователей и заказчика помогает снизить риски недоразумений, ошибок и неудачных решений в дальнейшем процессе разработки.

Основные этапы жизненного цикла разработки ПО:

2. Проектирование: На этом этапе разрабатывается архитектура программного продукта. Определяются компоненты, их взаимосвязи, структура базы данных, интерфейсы и другие аспекты системы.

Проектирование

Проектирование в контексте разработки программного обеспечения - это этап, на котором определяется общая структура, архитектура, компоненты и взаимодействие между ними для создания программного продукта, который удовлетворит заданным требованиям. Проектирование является ключевым этапом, так как он определяет фундаментальные решения, которые будут влиять на всю последующую разработку.

Этапы проектирования:

Определение архитектуры: На этом этапе создается общая архитектура системы, определяются компоненты, их взаимосвязи и взаимодействие. Решается, как система будет разбита на подсистемы и модули, какие паттерны архитектуры будут использованы.

Этапы проектирования:

Проектирование компонентов: Для каждого компонента определяются его функции, интерфейсы, методы и структура. Решается, какие данные будут храниться в компонентах, как они будут обрабатываться и передаваться.

Этапы проектирования:

Проектирование интерфейсов: Создание пользовательских и программных интерфейсов. Для пользовательского интерфейса определяется внешний вид, расположение элементов и взаимодействие с пользователем. Для программных интерфейсов определяются методы, параметры и способы взаимодействия с другими компонентами.

Этапы проектирования:

Выбор технологий: Решение о том, какие технологии, языки программирования, фреймворки и инструменты будут использованы для реализации компонентов и системы в целом.

Проектирование баз данных: Если система требует хранения данных, то на этом этапе определяется структура базы данных, таблицы, поля и связи между ними.

Этапы проектирования:

Управление данными и состоянием: Определение, как данные будут обрабатываться и храниться в системе. Решение о том, какие данные будут сохраняться в памяти, какие будут храниться в базе данных, какие будут передаваться между компонентами.

Разработка деталей: Подготовка детальных спецификаций и документации, описывающей работу компонентов, их функции, интерфейсы и способы взаимодействия.

Этапы проектирования:

Проверка и верификация проектирования: Проверка того, что проектирование соответствует требованиям и удовлетворяет целям проекта. Проверка на соответствие архитектурным принципам и паттернам.

Важно отметить, что хорошее проектирование помогает создать систему, которая будет более легко расширяться, поддерживаться и изменяться в будущем. Проектирование также помогает избежать некоторых проблем и ошибок, которые могут возникнуть на более поздних этапах разработки.

Основные этапы жизненного цикла разработки ПО:

3. Реализация (разработка): Этот этап связан с созданием и написанием кода программы на выбранном языке программирования. Программисты создают компоненты и функции, соответствующие спецификациям из предыдущих этапов.

Реализация (разработка)

Разработка программного обеспечения - это этап, на котором код программы создается и реализуются проектированные компоненты, функции и интерфейсы. Этот этап связан с тем, как программисты переводят концепции и архитектурные решения в рабочий и исполняемый код.

Этапы разработки:

Написание исходного кода: Программисты создают код, который реализует проектированные компоненты и функциональность. Они используют выбранный язык программирования и следуют стандартам и лучшим практикам для написания чистого, понятного и поддерживаемого кода.

Этапы разработки:

Тестирование компонентов: После написания каждого компонента или модуля проводится тестирование для проверки его функциональности. Это может включать в себя юнит-тестирование, где проверяется работа отдельных функций или методов.

Этапы разработки:

Интеграция компонентов: После того как компоненты протестированы индивидуально, они интегрируются вместе для проверки того, как они взаимодействуют друг с другом. Это может включать в себя интеграционное тестирование, чтобы убедиться, что компоненты корректно обмениваются данными и работают согласованно.

Этапы разработки:

Тестирование системы в целом: После интеграции всех компонентов проводится тестирование всей системы в целом. Это включает в себя проверку, что все компоненты работают вместе, а также функциональное и нефункциональное тестирование.

Этапы разработки:

Отладка и исправление ошибок: Если в процессе разработки выявляются ошибки или недоработки, программисты проводят отладку, чтобы найти и исправить проблемы.

Тестирование безопасности: Проводится тестирование на наличие уязвимостей и проверка безопасности системы, чтобы обеспечить защиту от взломов и утечек данных.

Этапы разработки:

Документирование кода: Код документируется с помощью комментариев и описаний, чтобы другие разработчики могли понять, как работает код и как использовать разработанные компоненты.

Этапы разработки:

Тестирование производительности: Если это требуется, проводится тестирование производительности, чтобы убедиться, что система работает достаточно быстро и эффективно.

Этапы разработки:

Непрерывная интеграция (CI) и непрерывная доставка (CD): Процессы автоматической сборки, тестирования и доставки помогают автоматизировать процесс разработки и обеспечить быструю и надежную поставку изменений в продукт.

Этапы разработки:

Разработка - это активный процесс создания кода, который выполняет требования и функции, определенные на предыдущих этапах анализа, проектирования и сбора требований. Этот этап требует не только технических навыков программирования, но и умения работать в команде, следовать стандартам кодирования и поддерживать хорошие практики разработки.

Основные этапы жизненного цикла разработки ПО:

4. Тестирование: Программное обеспечение подвергается различным видам тестирования, таким как юнит-тестирование, интеграционное тестирование и функциональное тестирование, чтобы обнаружить и устранить ошибки и недоработки.

Тестирование

Тестирование программного обеспечения - это процесс проверки и оценки работы программы с целью обнаружения ошибок, уверенности в соответствии требованиям и улучшения качества продукта. Тестирование играет ключевую роль в обеспечении надежности, функциональности и производительности программного продукта.

Этапы тестирования:

Планирование тестирования: Определение целей, объема и стратегии тестирования. Это включает в себя определение того, что будет тестироваться, какие тесты будут проводиться и как будет оцениваться успешность тестирования.

Этапы тестирования:

Создание тестовых случаев: Разработка тестовых сценариев, которые описывают, какие действия и входные данные будут использоваться для проверки определенных аспектов программы. Тестовые случаи могут включать в себя позитивные и негативные сценарии.

Этапы тестирования:

Выполнение тестов: Запуск тестовых сценариев на программе и сравнение результатов с ожидаемыми. Тесты могут включать автоматические и ручные тесты. Автоматические тесты выполняются с использованием специальных инструментов и фреймворков, ручные тесты проводятся тестировщиками вручную.

Этапы тестирования:

Анализ результатов: После выполнения тестов анализируются полученные результаты. Выявленные ошибки и недоработки документируются, их серьезность оценивается, и решается, какие из них следует исправить.

Исправление ошибок: Разработчики исправляют выявленные ошибки и недоработки, после чего тесты могут быть выполнены повторно, чтобы убедиться, что проблемы были успешно устранены.

Этапы тестирования:

Регрессионное тестирование: После внесения изменений тесты могут быть выполнены для уверенности в том, что исправления не привели к появлению новых ошибок.

Тестирование производительности: Проверка, как система работает под нагрузкой и в различных условиях. Это может включать тестирование скорости, надежности и масштабируемости.

Этапы тестирования:

Тестирование безопасности: Проверка на наличие уязвимостей и защиты от атак. Это особенно важно для систем, которые обрабатывают чувствительные данные.

Тестирование пользовательского интерфейса: Проверка удобства использования, соответствия дизайну и соответствия требованиям пользователя.

Этапы тестирования:

Документирование тестирования: Описание проведенных тестов, результатов и выявленных проблем. Эта документация может быть полезна для будущих обновлений и для обеспечения прозрачности процесса.

Тестирование помогает выявить ошибки и дефекты на ранних этапах разработки, что позволяет их исправить до выпуска продукта. Это способствует улучшению качества программного обеспечения и повышению удовлетворенности пользователей.

Основные этапы жизненного цикла разработки ПО:

5. Внедрение и развертывание: Завершив тестирование, программный продукт готов к внедрению в реальное окружение.

Это может включать в себя развертывание на серверах, установку на клиентских машинах и другие шаги, необходимые для запуска.

Внедрение и развертывание

Внедрение программного обеспечения - это последний этап в жизненном цикле разработки, на котором готовый продукт переносится из среды разработки в реальную среду пользователя. Этот этап включает в себя развертывание программы, обучение пользователей, обеспечение поддержки и следующие за выпуском действия.

Этапы внедрения:

Подготовка к развертыванию: Подготовка окружения для развертывания программы. Это может включать в себя установку и настройку необходимых серверов, баз данных, сетевых ресурсов и других компонентов.

Этапы внедрения:

Установка и развертывание: Программное обеспечение развертывается на целевых системах. Это может включать в себя установку на серверы, настройку баз данных, развертывание в облаке и т.д.

Этапы внедрения:

Интеграция и тестирование в реальной среде: После развертывания проводится тестирование программы в реальной среде, чтобы убедиться, что она работает корректно и соответствует ожиданиям пользователей. Это может включать функциональное и интеграционное тестирование, а также проверку производительности и безопасности.

Этапы внедрения:

Обучение пользователей: Пользователи получают обучение по использованию нового программного продукта. Это может включать в себя проведение семинаров, вебинаров, создание обучающей документации и другие методы.

Этапы внедрения:

Поддержка и сопровождение: После внедрения необходимо обеспечить поддержку пользователей и реагировать на возникающие вопросы и проблемы. Важно обеспечить каналы обратной связи и быстро реагировать на запросы.

Этапы внедрения:

Мониторинг и анализ: После внедрения следует мониторить работу программы в реальной среде, собирать данные о производительности и использовании, а также анализировать обратную связь пользователей.

Этапы внедрения:

Обновления и улучшения: После внедрения могут появляться обновления и улучшения продукта, основанные на обратной связи и новых требованиях. Эти изменения также должны быть развернуты и протестированы.

Этапы внедрения:

Оценка и обратная связь: После некоторого времени использования пользователи и разработчики могут провести оценку успешности внедрения, а также собрать обратную связь, чтобы выявить сильные и слабые стороны и внести коррективы.

Этапы внедрения:

Внедрение - это ключевой этап, так как он влияет на то, как успешно новое программное обеспечение будет принято пользователями и внедрено в рабочие процессы. Важно провести этот этап внимательно и организованно, чтобы минимизировать риски и обеспечить успешное внедрение продукта.

Основные этапы жизненного цикла разработки ПО:

6. Поддержка и обслуживание: После внедрения продукта начинается его эксплуатация. В этот период проводятся обновления, патчи, исправления ошибок и поддержка пользователей..

Поддержка и обслуживание

Поддержка программного обеспечения - это процесс обеспечения работоспособности, надежности и обновлений программного продукта после его внедрения. Этот этап важен для обеспечения долгосрочной успешной эксплуатации программы пользователем.

Поддержка включает в себя следующие аспекты:

Техническая поддержка: Пользователи могут сталкиваться с различными техническими проблемами, включая ошибки, неполадки, несовместимости и другие. Техническая поддержка предоставляет пользователю помощь в решении таких проблем. Обычно это происходит через онлайн-платформы, электронную почту, телефонные линии поддержки или чаты.

Поддержка включает в себя следующие аспекты:

Регулярные обновления: В процессе эксплуатации программного продукта могут выявляться ошибки, недоработки или требования к новой функциональности. Обновления помогают исправить ошибки, внести улучшения и обеспечить совместимость с новыми технологиями. Обновления могут быть плановыми (регулярные релизы) или внеплановыми (патчи для устранения критических проблем).

Поддержка включает в себя следующие аспекты:

Мониторинг и анализ: Поддержка включает в себя постоянный мониторинг работы программного продукта в реальной среде. Может использоваться мониторинг производительности, анализ журналов ошибок и сбоев, сбор статистики использования и другие методы для выявления потенциальных проблем.

Поддержка включает в себя следующие аспекты:

Управление безопасностью: Обеспечение безопасности программного продукта включает в себя реакцию на новые угрозы и уязвимости. Поддержка включает обновления, которые вносят патчи для устранения уязвимостей и обеспечивают защиту от новых угроз.

Поддержка включает в себя следующие аспекты:

Поддержка пользователей: Оказание помощи пользователям в различных вопросах, связанных с использованием программного продукта. Это может быть связано с объяснением функциональности, демонстрацией возможностей, решением проблем с настройками и т.д.

Поддержка включает в себя следующие аспекты:

Обратная связь и взаимодействие: Поддержка предполагает взаимодействие с пользователями, сбор обратной связи, а также анализ запросов и предложений для будущих обновлений и улучшений.

Поддержка включает в себя следующие аспекты:

Документация: Поддержка включает в себя создание и обновление документации для пользователя, которая может включать инструкции по установке, использованию, решению проблем и другую полезную информацию.

Поддержка включает в себя следующие аспекты:

Обучение: Поддержка включает обучение пользователей новым функциям или изменениям, внесенным в обновлениях.

Решение проблем и ошибок: В случае обнаружения ошибок или неполадок пользователи обращаются в службу поддержки, и их проблемы решаются в согласованные сроки.

Поддержка включает в себя следующие аспекты:

Поддержка помогает убедиться, что продукт остается актуальным, надежным и полезным для пользователей на протяжении всего его жизненного цикла. Это важный аспект в обеспечении удовлетворенности пользователей и успешной эксплуатации программного обеспечения.

Основные этапы жизненного цикла разработки ПО:

7. Завершение или вывод из эксплуатации: Если продукт устарел или перестал быть полезным, его можно вывести из эксплуатации.

Завершение или вывод из эксплуатации

На стадии вывода программного продукта из эксплуатации прекращаются процессы сопровождения ПП, проводится изъятие из эксплуатации программного продукта и связанных с ним программных подсистем.

Все связанные с выводимым из эксплуатации программным продуктом подсистемы, документы и данные должны быть помещены в архивы.

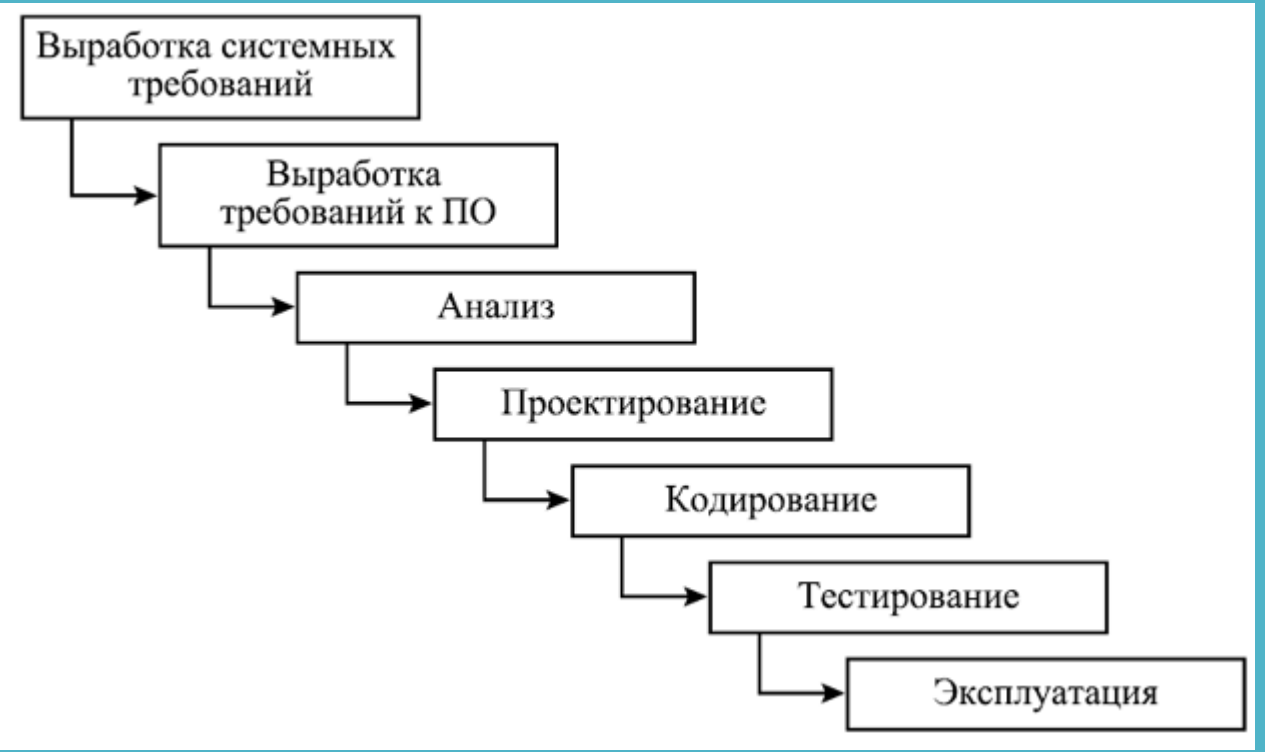
Жизненный цикл разработки ПО может различаться в зависимости от модели разработки (например, каскадная модель, спиральная модель, Agile и др.) и конкретных требований проекта.

3. Модели разработки ПО

Моделей разработки ПО много, но, в общем случае, классическими можно считать каскадную, итерационную, инкрементальную, спиральную и гибкую.

Каскадная (водопадная) модель

Сейчас, представляет, скорее, исторический интерес, т.к. в современных проектах практически не применима. Она предполагает однократное выполнение каждой из фаз проекта, которые, в свою очередь, строго следуют друг за другом.



Особенности каскадной модели:

-высокий уровень формализации процессов;

-большое количество документации;

-жесткая последовательность этапов жизненного цикла без возможности возврата на предыдущий этап.

Минусы:

Избыточная документация.

Очень не гибкая методология.

Может создать ошибочное впечатление о работе над проектом (например, фраза «45% выполнено» не несёт за собой никакой полезной информации, а является всего лишь инструментов для менеджера проекта).

У заказчика нет возможности ознакомиться с системой заранее и даже с «Пилотом» системы.

У пользователя нет возможности привыкать к продукту постепенно.

Все требования должны быть известны в начале жизненного цикла проекта.

Возникает необходимость в жёстком управлении и регулярном контроле, иначе проект быстро выбьется из графиков.

Отсутствует возможность учесть переделку, весь проект делается за один раз.

Плюсы:

Высокая прозрачность разработки и фаз проекта.

Чёткая последовательность.

Стабильность требований.

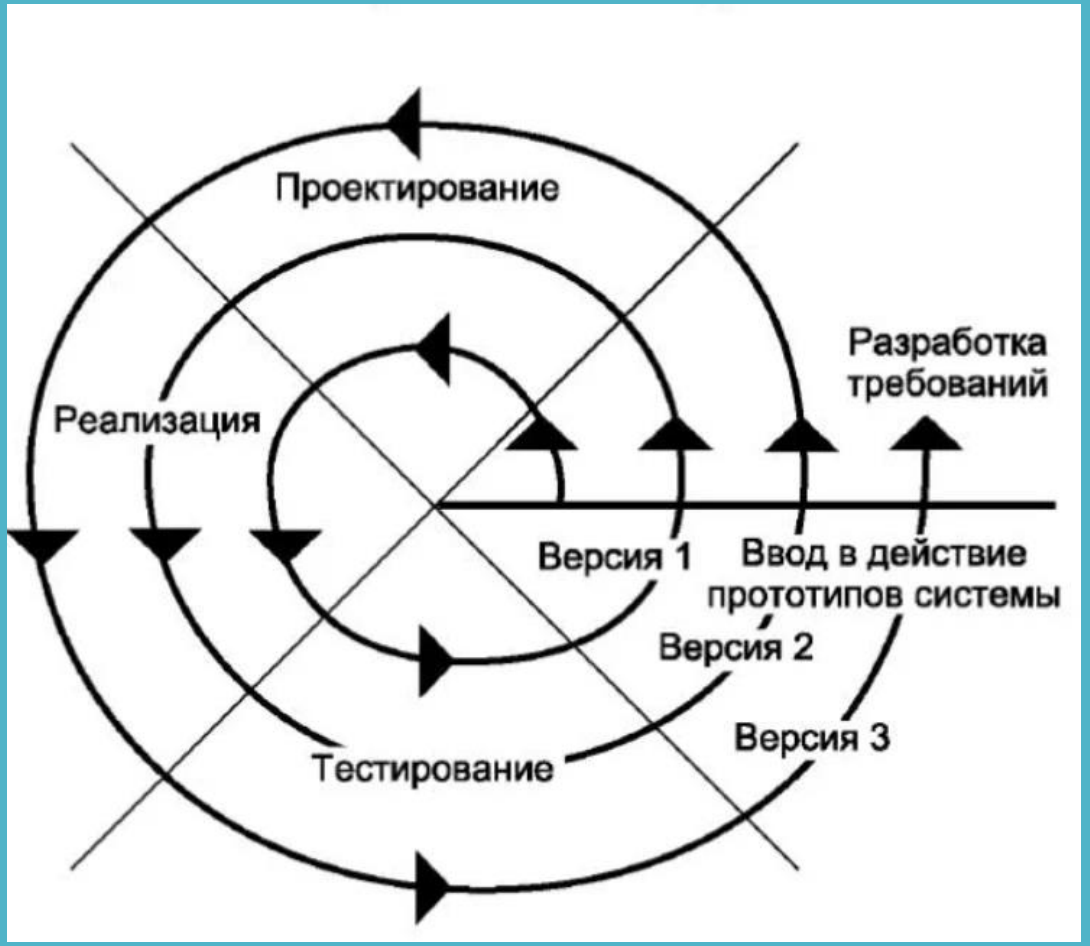
Строгий контроль менеджмента проекта.

Облегчает работу по составлению плана проекта и сбора команды проекта.

Хорошо определяет процедуру по контролю качества.

Модель спирального развития

Это эволюционная модель жизненного цикла представляет собой цикл, в котором каждая итерация представляет собой "спираль", чередующуюся с этапами анализа, разработки и тестирования.



Преимущества модели спирального развития:

Гибкость: Позволяет быстро адаптироваться к изменениям требований и рискам.

Управление рисками: Акцент на рисках позволяет предвидеть и снижать возможные проблемы.

Инкрементальное развитие: Постепенное улучшение продукта на каждой итерации.

Недостатки модели спирального развития:

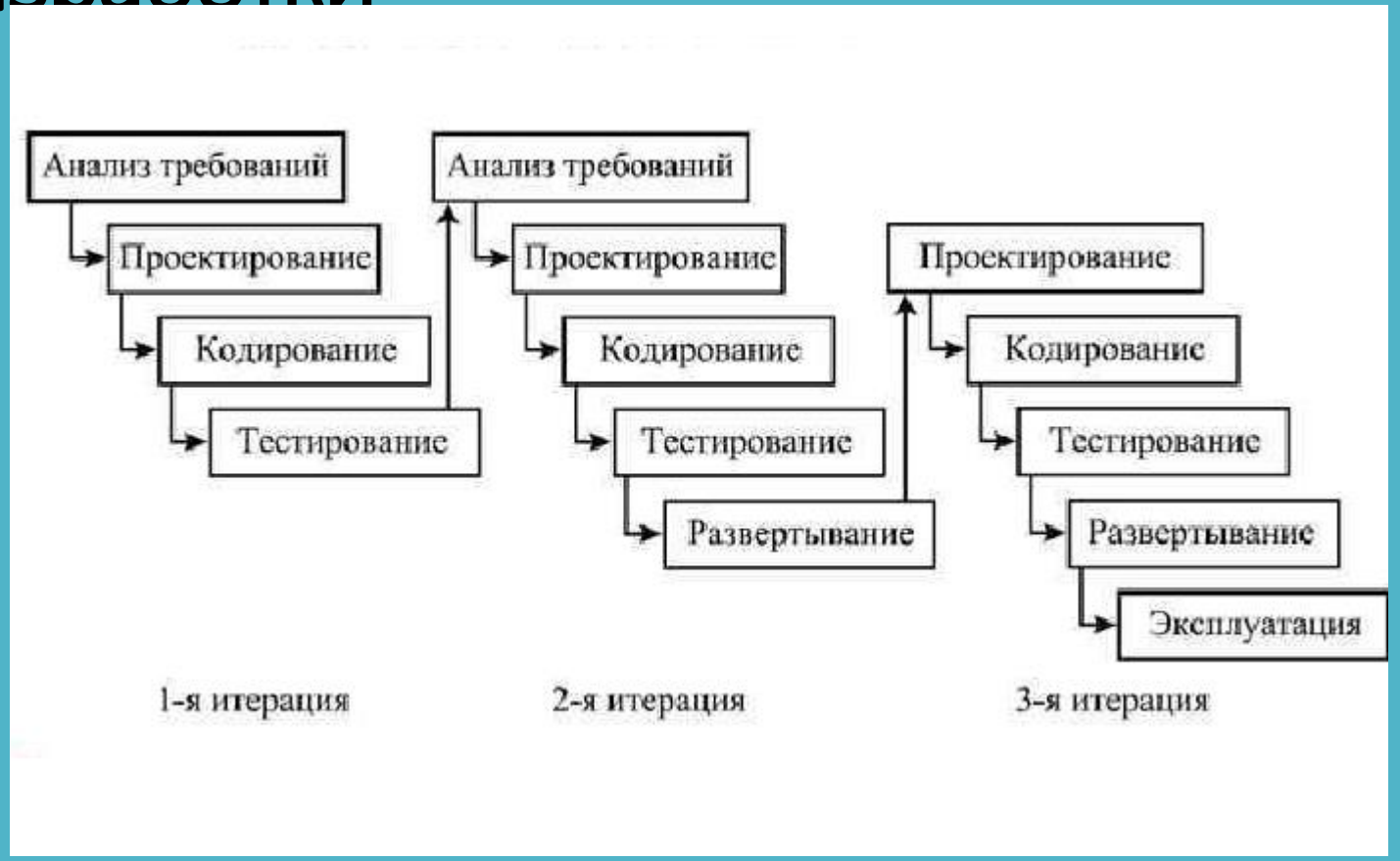
Сложность управления: Постоянные итерации и управление рисками требуют хорошего планирования и координации.

Возможность затяжных циклов: Несмотря на гибкость, если не проводить достаточно четкой оценки и планирования, процесс разработки может затянуться.

Модель спирального развития подходит для проектов, где высока степень неопределенности, изменения требований, а также важно управление рисками.

Итеративная модель разработки

В итеративных моделях разработки проект разбивается на серию коротких циклов, называемых итерациями. Каждая итерация представляет собой полный жизненный цикл разработки, включая анализ, проектирование, разработку, тестирование и внедрение. После завершения каждой итерации получается работающий продукт, который может быть продемонстрирован заказчику для получения обратной связи. Затем итерация повторяется, причем каждая следующая итерация уточняет и расширяет функциональность, исходя из обратной связи.



Инкрементальная модель

В инкрементальных моделях процесс разработки разбивается на набор инкрементов, каждый из которых добавляет к предыдущему версии новую функциональность или улучшения. Каждый инкремент представляет собой полный цикл разработки, от анализа до внедрения. По завершении каждого инкремента получается версия продукта с дополнительной функциональностью.

Преимущества итеративных и инкрементальных моделей:

Гибкость: Они позволяют быстро адаптироваться к изменениям требований и обратной связи заказчика.

Быстрые результаты: Каждый цикл или инкремент приносит результат, что полезно для оценки прогресса и демонстрации заказчику.

Улучшенная обратная связь: Заказчик может видеть и оценить части продукта на ранних этапах.

Постепенное развитие: Функциональность развивается постепенно, что позволяет избегать долгих периодов без видимого прогресса.

Недостатки:

Сложность управления: Необходимо тщательно планировать и координировать каждый цикл или инкремент, чтобы избежать хаоса.

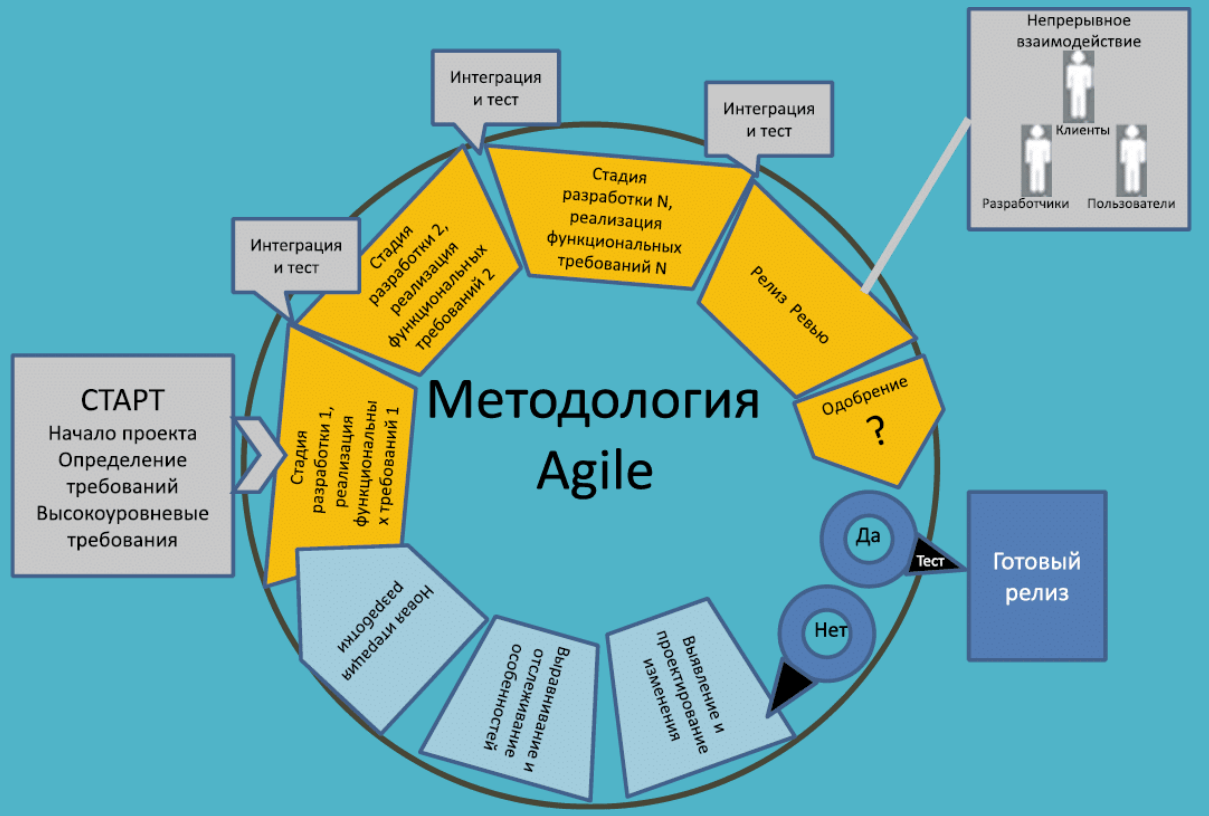
Возможность недооценки: Не всегда можно точно оценить объем работы на каждом этапе, что может привести к неожиданным трудностям.

Примерами итеративных и инкрементальных методологий являются Agile, Scrum, Extreme Programming (XP) и RUP (Rational Unified Process).

Гибкая методология разработки (Agile)

Это набор подходов и принципов, направленных на организацию и управление разработкой программного обеспечения с акцентом на гибкость, сотрудничество, адаптивность к изменениям и обеспечение быстрых итераций для достижения максимальной ценности для заказчика.

Agile уделяет особое внимание командной работе, обратной связи заказчика и постоянной адаптации к изменяющимся требованиям.



Основные принципы Agile:

Люди и взаимодействие важнее процессов и инструментов: Сотрудничество и коммуникация между членами команды и с заказчиком приоритетнее формальных процессов и инструментов.

Работающий продукт важнее исчерпывающей документации: Главная цель - получить работающий и полезный продукт, а не создать большое количество документации.

Сотрудничество с заказчиком важнее согласования условий контракта: Взаимодействие с заказчиком и его обратная связь ориентируют на разработку наиболее полезных функций.

Готовность к изменениям важнее следования плану: Agile признает, что требования могут изменяться, и команда должна готово реагировать на эти изменения.

Основные методологии Agile:

Scrum: Один из наиболее популярных фреймворков Agile. Работа организуется в коротких циклах, называемых спринтами. Каждый спринт обычно длится от 2 до 4 недель и заканчивается демонстрацией готового функционала.

Extreme Programming (XP): Уделяет большое внимание техническим практикам и качеству кода. Включает в себя практики, такие как парное программирование, тестирование на уровне кода и частые релизы.

Kanban: Управление рабочим процессом с помощью визуальных досок, на которых задачи перемещаются между этапами, что обеспечивает прозрачность и оптимизацию процесса.

Lean: Ориентирована на устранение "потерь" в процессе разработки и достижение максимальной ценности для заказчика.

Преимущества Agile:

Гибкость: Может быстро адаптироваться к изменениям требований и обратной связи.

Близкое взаимодействие с заказчиком: Обратная связь заказчика ценна на всех этапах разработки.

Рабочий продукт на каждой итерации: Заказчик видит результаты в работающем состоянии на ранних этапах.

Недостатки:

Может быть сложно применить в больших организациях с жесткими процессами.

Требует высокой организованности и дисциплины в команде.

Agile подходит для проектов с меняющимися требованиями и когда заказчику важно видеть результаты на ранних этапах.

Выбор подходящей модели в зависимости от проекта.

Выбор подходящей модели разработки программного обеспечения зависит от множества факторов, включая характеристики проекта, требования заказчика, команду разработчиков, сроки и бюджет. Важно выбрать модель, которая наилучшим образом соответствует конкретным условиям и потребностям проекта.

Некоторые факторы, которые следует учитывать при выборе модели:

Требования к проекту: Если требования к проекту стабильны и хорошо определены заранее, то каскадная модель или инкрементальная модель могут подойти. Если требования меняются часто, то Agile-методологии, такие как Scrum или Kanban, могут быть более подходящими.

Некоторые факторы, которые следует учитывать при выборе модели:

Размер и сложность проекта: Для маленьких и средних проектов, где требования ясны, каскадная или инкрементальная модели могут быть хорошим выбором. Для больших и сложных проектов, где необходимо частое взаимодействие с заказчиком и адаптация к изменениям, Agile-методологии могут быть более подходящими.

Некоторые факторы, которые следует учитывать при выборе модели:

Сроки: Если проект имеет жесткие сроки и нельзя себе позволить частые итерации, то каскадная модель или спиральная модель могут подойти. Если есть возможность разбивать проект на короткие циклы, то Agile-методологии подходят лучше.

Некоторые факторы, которые следует учитывать при выборе модели:

Бюджет: Некоторые модели могут потребовать больше времени и ресурсов, например, каскадная модель. Agile-методологии могут быть более эффективными с точки зрения бюджета, так как позволяют рано видеть результаты.

Некоторые факторы, которые следует учитывать при выборе модели:

Опыт команды: Если команда имеет опыт работы с определенной моделью, то это может повлиять на выбор. Например, если команда хорошо знакома с Agile, то это может быть предпочтительным выбором.

Некоторые факторы, которые следует учитывать при выборе модели:

Уровень рисков: Если проект имеет высокий уровень рисков, то модели, которые акцентируют управление рисками, например, спиральная модель, могут быть более подходящими.

Требования к коммуникации: Если важно поддерживать постоянное взаимодействие с заказчиком и коммуникацию внутри команды, то Agile-методологии, такие как Scrum, могут обеспечить это.

П.С: Важно подбирать модель, которая наиболее эффективно соответствует уникальным характеристикам проекта и потребностям команды. При этом иногда может быть полезным комбинировать элементы разных моделей в зависимости от конкретных обстоятельств.

Домашнее задание:

1.Выберите две любые методологии разработки программного обеспечения.

2.Сравните выбранные методологии между собой и ответьте на следующие вопросы:

- в чем заключаются основные различия между этапами жизненного цикла разработки ПО в каждой из методологий?
- какие преимущества и недостатки имеют выбранные методологии?
- какие типы проектов лучше подходят для каждой из методологий?

3.Подготовьте краткий отчет (примерно 1 стр.), включающий сравнение методологий и выводы.

П.С: Разместите отчет (в формате PDF) в своем репозитории “HomeWork/lesson1/report.pdf”.

Ссылку отправить на почту: colledge20education23@gmail.com