1. История и причины возникновения программной инженерии и её особенности.

a. Понятие инженерии и инженерных технологий. Типы инженерий.

a. Понятие инженерии и инженерных технологий

Инженерия – прикладная наука, использующая знания математики, физики и других дисциплин для проектирования, создания и оптимизации технических систем, конструкций и процессов.

Инженерные технологии – совокупность методов, инструментов и практик, применяемых для реализации инженерных решений (например, CAD-моделирование, автоматизированное тестирование, методы машинного обучения).

Типы инженерий (основные направления):

• Механическая – машины, механизмы, робототехника.

• Электротехническая – электроника, энергетика, микропроцессоры.

• Гражданская – строительство зданий, мостов, инфраструктуры.

• Химическая – процессы переработки, создание материалов.

• Компьютерная (программная) – разработка ПО, алгоритмов, ИИ.

• Аэрокосмическая – самолёты, ракеты, спутники.

• Биомедицинская – медицинские приборы, биотехнологии.

• Промышленная – оптимизация производства.

• Экологическая – защита окружающей среды.

• Нефтегазовая – добыча и переработка ресурсов.

Программная инженерия – частный случай инженерии, специализирующийся на создании и управлении качественным ПО.

b. История и причины возникновения программной инженерии.

1. Предпосылки (1940–1960-е)

Первые программы создавались вручную для ЭВМ (например, ENIAC, 1945).

Отсутствие стандартов: код писали математики и физики без системного подхода.

Проблемы: ошибки, сложность сопровождения, высокая стоимость разработки.

2. «Кризис программного обеспечения» (1960–1970-е)

Рост сложности ПО (например, OS/360 для IBM).

Проблемы:

Проекты выходили за сроки и бюджет.

Низкая надёжность программ.

Трудности масштабирования.

Решение: необходимость дисциплины, аналогичной традиционной инженерии.

3. Появление программной инженерии (1968)

Термин «software engineering» введён на конференции НАТО (1968) для обозначения системного подхода к разработке ПО.

Ключевые идеи:

Применение инженерных принципов (планирование, документирование, тестирование).

Стандартизация процессов.

4. Развитие (1980–2000-е)

Объектно-ориентированное программирование (ООП) – улучшение структуры кода.

Гибкие методологии (Agile, Scrum) – ответ на неэффективность «водопадной» модели

Автоматизация: CI/CD, DevOps.

5. Современность (2020-е)

ИИ в разработке (GitHub Copilot).

Микросервисы и облачные технологии.

Основные причины возникновения:

Сложность ПО – рост масштабов проектов.

Ненадёжность – частые сбои и ошибки.

Высокая стоимость – необходимость оптимизации процессов.

Необходимость стандартов – для командной работы и поддержки.

Итог: Программная инженерия сформировалась как ответ на хаотичность ранней разработки ПО, чтобы сделать её предсказуемой, управляемой и экономически эффективной

c. Направления программной инженерии.

1. По этапам жизненного цикла ПО

Системный анализ и проектирование

Проектирование архитектуры ПО (микросервисы, монолиты).

Работа с требованиями (SRS, use cases).

Разработка (Implementation)

Написание кода на различных языках (Python, Java, C++, etc.).

Применение парадигм (ООП, функциональное программирование).

Тестирование (QA)

Модульное, интеграционное, системное тестирование.

Автоматизированное тестирование (Selenium, JUnit).

Сопровождение и эволюция

Рефакторинг, исправление багов, обновления.

2. По специализациям в разработке

Frontend-разработка

Интерфейсы (HTML/CSS/JavaScript, React, Vue, Angular).

Backend-разработка

Серверная логика (Node.js, Django, Spring, Golang).

Fullstack-разработка

Комбинирование frontend и backend.

Embedded-разработка

Программирование микроконтроллеров (C, Rust, Arduino).

Мобильная разработка

iOS (Swift), Android (Kotlin), кроссплатформенные решения (Flutter).

GameDev

Разработка игр (Unity, Unreal Engine, Godot).

3. По работе с данными

Data Engineering

Обработка больших данных (Hadoop, Spark, Kafka)

ETL-процессы (Extract, Transform, Load).

Machine Learning Engineering

Разработка и внедрение моделей ИИ (Python, TensorFlow, PyTorch).

Database Engineering

Проектирование БД (SQL, NoSQL), оптимизация запросов.

4. По управлению и автоматизации

DevOps & SRE

Автоматизация развертывания (Docker, Kubernetes).

Мониторинг и надежность систем (Prometheus, Grafana).

Cloud Engineering

Работа с облачными платформами (AWS, Azure, GCP).

Security Engineering

Кибербезопасность, пентестинг (OWASP, Burp Suite).

5. По методологиям и процессам

Agile & Scrum

Гибкие методологии управления проектами.

Software Architecture

Проектирование масштабируемых систем (DDD, Clean Architecture)

Project Management

Управление командами (Jira, Trello).

d. Этапы развития программной инженерии

1. Ранний период (1940–1950-е) – Ручное программирование

Характеристика:

Программы писались в машинных кодах и ассемблере.

Отсутствие языков высокого уровня.

Проблемы:

Крайне низкая производительность разработки.

Ошибки исправлялись физическим переключением проводов (ENIAC).

Ключевые события:

Создание первых ЭВМ (ENIAC, 1945; UNIVAC, 1951).

2. Эра языков высокого уровня (1950–1960-е) – Фортранизация

Характеристика:

Появление языков Fortran (1957), COBOL (1959), ALGOL (1958).

Программирование стало ближе к математике, а не к аппаратуре.

Проблемы:

Рост сложности ПО → трудно поддерживать код.

Нет стандартов разработки.

Ключевые события:

Разделение на системных и прикладных программистов.

3. Кризис программного обеспечения (1960–1970-е) – Осознание проблем

Характеристика:

Проекты выходят за сроки и бюджет (например, OS/360 от IBM).

ПО становится слишком сложным для ad-hoc разработки.

Проблемы:

Нет управления требованиями.

Катастрофические сбои (например, ошибки в ПО для космических миссий).

Ключевые события:

1968 г. – Конференция НАТО, где введён термин "software engineering".

Появление структурного программирования (Дейкстра, 1968).

4. Становление инженерных методов (1970–1980-е) – Процессы и стандарты

Характеристика:

Водопадная модель (Royce, 1970).

Появление реляционных баз данных (SQL, 1974).

Проблемы:

Жёсткость waterfall → сложно вносить изменения.

Ключевые события:

Развитие модульного программирования.

Первые CASE-инструменты (Computer-Aided Software Engineering).

5. Объектно-ориентированная революция (1980–1990-е) – ООП и GUI

Характеристика:

Широкое внедрение C++ (1985), Java (1995).

Появление графических интерфейсов (Windows, MacOS).

Проблемы:

Усложнение архитектуры.

Ключевые события:

1986 г. – Первая версия UML (унифицированный язык моделирования).

Развитие клиент-серверных приложений.

6. Гибкие методологии (1990–2000-е) – Agile и Open Source

Характеристика:

Манифест Agile (2001) → Scrum, Kanban, XP.

Расцвет Open Source (Linux, Apache, Python).

Проблемы:

Управление распределёнными командами.

Ключевые события:

Появление Git (2005).

Развитие веб-разработки (PHP, JavaScript, Ruby on Rails).

7. Современная эра (2010–2020-е) – DevOps, Cloud и AI

Характеристика:

DevOps – автоматизация CI/CD (Docker, Kubernetes).

Облачные технологии (AWS, Azure, GCP).

Машинное обучение в разработке (GitHub Copilot).

Проблемы:

Безопасность и масштабируемость.

Ключевые события:

Микросервисная архитектура.

Low-code/no-code платформы.

2. Области знаний программной инженерии

a. Направления знаний в области программной инженерии: классификация и состав по направлениям

Программная инженерия охватывает множество направлений, которые можно классифицировать следующим образом:

Теоретические основы

Алгоритмы и структуры данных.

Теория вычислений и сложности.

Математическая логика и формальные методы.

Процессы разработки ПО

Модели жизненного цикла (Waterfall, Agile, DevOps).

Управление проектами (Scrum, Kanban).

Методы оценки качества (CMMI, ISO 12207).

Методы и технологии программирования

Парадигмы программирования (ООП, функциональное, процедурное).

Языки программирования (Python, Java, C++, Rust).

Фреймворки и библиотеки (React, Spring, TensorFlow).

Архитектура ПО

Паттерны проектирования (MVC, Microservices, Event-Driven).

Масштабируемость и отказоустойчивость.

Тестирование и обеспечение качества

Виды тестирования (unit, integration, system).

Автоматизированное тестирование (Selenium, JUnit).

Управление конфигурацией и DevOps

Системы контроля версий (Git).

CI/CD (Jenkins, GitHub Actions).

Контейнеризация (Docker, Kubernetes).

Безопасность ПО

Криптография.

Пентестинг и анализ уязвимостей.

Специализированные области

Data Engineering (ETL, Big Data).

Machine Learning Engineering.

Разработка встроенных систем (Embedded).

b. Кодекс этики программного инженера – назначение, содержание и значение

Назначение:

Определение моральных и профессиональных стандартов для разработчиков.

Обеспечение доверия к программным продуктам и их создателям.

Содержание (основные принципы):

Общественная ответственность

ПО не должно вредить людям или обществу.

Профессиональная компетентность

Инженер должен поддерживать и повышать свои знания.

Честность и прозрачность

Открытость в оценке рисков и ограничений ПО.

Уважение к интеллектуальной собственности

Запрет на плагиат и несанкционированное использование кода.

Конфиденциальность

Защита данных пользователей.

Значение:

Повышает качество и надежность ПО.

Снижает риски юридических и репутационных потерь.

Формирует доверие среди клиентов и коллег.

c. Основные понятия программной инженерии: артефакты, методы и модели

1. Артефакты (результаты разработки):

Исходный код – написанные программистом инструкции.

Документация – ТЗ, руководства пользователя, API-документация.

Исполняемые файлы – скомпилированные программы.

Модели и диаграммы – UML-схемы, ER-диаграммы.

2. Методы:

Структурное программирование – модульность, минимизация GOTO.

Объектно-ориентированное проектирование – инкапсуляция, наследование, полиморфизм.

Agile-методы – итеративная разработка, Scrum.

Формальные методы – математическая верификация кода.

3. Модели:

Waterfall – последовательные этапы (анализ → разработка → тестирование).

Spiral – итерации с оценкой рисков.

V-модель – связь тестирования с этапами разработки.

Гибкие модели (Agile, Scrum) – адаптивное планирование.

3. Основы деятельности по стандартизации в области программной инженерии

a. Объективная необходимость международной стандартизации в инженерных науках. Международные организации – регуляторы в области стандартизации

Необходимость стандартизации:

Обеспечение совместимости – единые стандарты позволяют разным системам и ПО взаимодействовать.

Повышение качества – стандарты задают требования к надежности, безопасности и производительности.

Снижение затрат – унификация процессов уменьшает издержки разработки и поддержки.

Международное сотрудничество – общие нормы упрощают выход на глобальные рынки.

Международные организации-регуляторы:

ISO (International Organization for Standardization) – разрабатывает стандарты ISO/IEC для ПО (например, ISO/IEC 12207).

IEC (International Electrotechnical Commission) – отвечает за стандарты в электротехнике и IT.

IEEE (Institute of Electrical and Electronics Engineers) – стандарты в области программной инженерии (например, IEEE 730 для управления качеством ПО).

ITU (International Telecommunication Union) – нормы для телекоммуникаций и связанного ПО.

b. Участники разработок международных стандартов в области программной инженерии и основные направления их деятельности

Участники:

Государственные институты – национальные органы стандартизации (например, ANSI в США, ГОСТ в России).

Корпорации и IT-компании – Microsoft, IBM, Google участвуют в создании стандартов (например, для облачных технологий).

Научные и профессиональные сообщества – ACM, IEEE Computer Society.

Открытые сообщества – консорциумы типа W3C (стандарты веб-технологий).

Направления деятельности:

Разработка стандартов жизненного цикла ПО (ISO/IEC 12207).

Стандартизация языков программирования (например, стандарты C++ от ISO).

Нормы для кибербезопасности (ISO/IEC 27001).

Рекомендации для DevOps и Agile (ISO/IEC 26515).

c. Классификация стандартов в области программной инженерии по уровню их применения. Иерархия структур, разрабатывающих стандарты в России

Классификация стандартов:

Международные (ISO, IEC, IEEE) – применяются глобально (например, ISO/IEC 25010 для качества ПО).

Национальные (ГОСТ, ANSI, DIN) – адаптация международных стандартов под местные требования.

Пример: ГОСТ Р ИСО/МЭК 12207 – российская версия ISO/IEC 12207.

Отраслевые – стандарты для конкретных сфер (например, медицинское ПО).

Корпоративные – внутренние стандарты компаний (например, Google Style Guide).

Иерархия структур в России:

Росстандарт – главный орган, утверждающий национальные стандарты (ГОСТ).

Технические комитеты (ТК) – разрабатывают стандарты в конкретных областях:

ТК 22 «Программная инженерия».

ТК 362 «Искусственный интеллект».

Профильные институты – например, ФСТЭК (стандарты безопасности).

Отраслевые ассоциации – РУССОФТ, АПКИТ.

4. Основные направления стандартизации в области программной инженерии

a. Классификация направлений стандартов

Стандарты в программной инженерии можно классифицировать по следующим направлениям:

Жизненный цикл ПО

Управление процессами разработки (ISO/IEC 12207).

Модели зрелости (CMMI, ISO/IEC 15504).

Качество ПО

Модели качества (ISO/IEC 25010).

Метрики и оценка качества.

Безопасность и защита данных

Стандарты информационной безопасности (ISO/IEC 27001).

Криптографические методы (ISO/IEC 18033).

Архитектура и проектирование

UML (ISO/IEC 19505).

Паттерны проектирования.

Тестирование

Процессы тестирования (ISO/IEC/IEEE 29119).

Автоматизация тестирования.

Управление конфигурацией

Системы контроля версий.

CI/CD (непрерывная интеграция и доставка).

Документирование

Стандарты технической документации (IEEE 830).

Специализированные области

Встраиваемые системы (AUTOSAR).

Искусственный интеллект и машинное обучение.

b. Содержание стандартов по основным направлениям

ISO/IEC 12207 (Жизненный цикл ПО)

Определяет процессы разработки, эксплуатации и сопровождения ПО.

Включает планирование, анализ требований, проектирование, тестирование.

ISO/IEC 25010 (Качество ПО)

Характеристики качества: функциональность, надежность, производительность, безопасность.

Модель заменяет ISO/IEC 9126.

ISO/IEC 27001 (Информационная безопасность)

Требования к системе управления безопасностью информации (СУБИ).

Включает оценку рисков и меры защиты.

ISO/IEC/IEEE 29119 (Тестирование ПО)

Процессы тестирования: планирование, проектирование, исполнение.

Определяет уровни тестирования (модульное, интеграционное, системное).

IEEE 830 (Документирование требований)

Рекомендации по написанию SRS (Software Requirements Specification).

Структура и содержание документации.

AUTOSAR (Автомобильные системы)

Стандарт для встраиваемого ПО в автомобильной промышленности.

Определяет архитектуру и интерфейсы.

c. Актуальные направления стандартизации в программной инженерии в последнее время (примерно в последние 5-10 лет)

DevOps и CI/CD

Стандарты для автоматизации развертывания и мониторинга (например, ISO/IEC 26515).

Интеграция с облачными платформами.

Искусственный интеллект и машинное обучение

Стандарты для этики ИИ (IEEE 7000).

Обеспечение качества моделей ML (ISO/IEC 23053).

Кибербезопасность и приватность

GDPR-совместимые стандарты обработки данных.

Безопасность IoT (ISO/IEC 30141).

Микросервисы и облачные технологии

Стандарты для контейнеризации (Kubernetes, Docker).

API-стандарты (OpenAPI, GraphQL).

Low-code/No-code разработка

Стандарты для визуального программирования.

Блокчейн и распределенные системы

Стандарты для смарт-контрактов (ISO/TC 307).

5. Документация в программной инженерии

a. Общая классификация документации в программной инженерии по всем этапам разработки программного проекта

Документация в программной инженерии классифицируется по этапам жизненного цикла ПО:

Документация этапа планирования:

Техническое задание (ТЗ)

Бизнес-план проекта

План управления проектом

Документация этапа анализа требований:

Software Requirements Specification (SRS)

Use Case-диаграммы

Пользовательские истории (User Stories)

Документация проектирования:

Software Design Document (SDD)

Диаграммы UML

Архитектурные решения

Прототипы интерфейсов

Документация разработки:

Технические спецификации

API-документация

Комментарии в коде

Руководство разработчика

Документация тестирования:

План тестирования

Тест-кейсы

Отчеты о тестировании

Отчеты об ошибках

Документация внедрения:

Руководство по установке

Руководство администратора

Миграционные скрипты

Эксплуатационная документация:

Руководство пользователя

Справочная система

Документация API (для внешних разработчиков)

Документация сопровождения:

Журнал изменений

Отчеты об инцидентах

Документация обновлений

b. Современные представления о масштабах программного проекта

Современная классификация масштабов проектов:

По размеру команды:

Индивидуальный проект (1 разработчик)

Малая команда (2-5 человек)

Средняя команда (6-20 человек)

Крупная команда (20+ человек)

Распределенные команды (географически распределенные)

По объему кода:

Мини-проекты (<1K строк кода)

Малые проекты (1K-10K строк)

Средние проекты (10K-100K строк)

Крупные проекты (100K-1M строк)

Очень крупные проекты (>1M строк)

По сложности:

Простые (линейные алгоритмы)

Средней сложности (несколько модулей)

Сложные (распределенные системы)

Очень сложные (искусственный интеллект, системы реального времени)

По длительности:

Краткосрочные (<3 месяцев)

Среднесрочные (3-12 месяцев)

Долгосрочные (1-3 года)

Очень долгосрочные (>3 лет)

c. Стандарты в области документирования процессов и артефактов программной инженерии

Основные международные стандарты документации:

IEEE 829 - Стандарт для тестовой документации

IEEE 1016 - Стандарт для документирования проектирования ПО

IEEE 830 - Стандарт для спецификации требований к ПО

ISO/IEC/IEEE 26515 - Стандарт для пользовательской документации

ISO/IEC 26531 - Стандарт для управления документацией

MIL-STD-498 - Военный стандарт документации (устарел, но влиятелен)

AUTOSAR - Стандарт документации для автомобильного ПО

IEC 62304 - Стандарт документации для медицинского ПО

d. Состав и содержание документации программного обеспечения

Основные виды документации и их содержание:

Техническая документация:

Техническое задание (цели, требования, ограничения)

Спецификация требований (функциональные и нефункциональные требования)

Архитектурное описание (диаграммы компонентов, последовательностей)

API-документация (эндпоинты, параметры, примеры запросов)

Пользовательская документация:

Руководство пользователя (пошаговые инструкции)

Справочное руководство (описание функций)

Руководство по установке

Руководство по миграции

Административная документация:

Руководство администратора

Документация по настройке

Документация по безопасности

Разработческая документация:

Комментарии в коде

Руководство по стилю кодирования

Описание алгоритмов

Документация модулей

Тестовая документация:

План тестирования

Тест-кейсы

Отчеты о тестировании

Метрики качества

Проектная документация:

План проекта

Отчеты о статусе

Протоколы совещаний

Отчеты о рисках

Примечание: В современных agile-проектах документация часто ведется в виде:

Wiki-страниц

Интерактивной документации (Swagger для API)

Комментариев в системе контроля версий

Онлайн-руководств с поиском

6. Методы и модели в области программной инженерии

a. Общая характеристика и содержание методов программной инженерии

i. Методы структурного анализа

Характеристика:

Основаны на декомпозиции системы на функциональные модули.

Используют иерархические диаграммы для представления структуры.

Основные техники:

DFD (Data Flow Diagram) – отображает потоки данных между процессами.

SADT (Structured Analysis and Design Technique) – моделирует функции и их взаимодействие.

Структурные схемы – иерархия модулей (например, диаграммы Хартли).

Применение:

Анализ требований к ПО.

Проектирование систем с четкими функциональными границами.

ii. Метод "сущность–связь" (ER-моделирование)

Характеристика:

Используется для проектирования баз данных.

Описывает данные в виде сущностей, их атрибутов и связей между ними.

Основные элементы:

Сущности (например, "Пользователь", "Заказ").

Атрибуты (свойства сущностей, например, "имя", "дата").

Связи (отношения между сущностями, например, "один-ко-многим").

Диаграммы:

ER-диаграммы (нотация Чена, Crow’s Foot).

Применение:

Проектирование реляционных БД.

Анализ предметной области.

iii. Метод объектно-ориентированного анализа (ООА)

Характеристика:

Основан на концепциях объектов, классов и их взаимодействия.

Использует инкапсуляцию, наследование и полиморфизм.

Основные техники:

UML (Unified Modeling Language) – стандарт для визуализации:

Диаграммы классов (Class Diagrams).

Диаграммы последовательностей (Sequence Diagrams).

Диаграммы вариантов использования (Use Case Diagrams).

Паттерны проектирования (например, MVC, Singleton).

Применение:

Разработка сложных систем с высокой степенью повторного использования кода.

Agile-проекты с итеративным проектированием.

b. Модель жизненного цикла программного продукта: этапы развития, назначение и использование

Этапы развития моделей ЖЦ:

Каскадная (Waterfall, 1970-е) – линейные этапы (анализ → проектирование → разработка → тестирование → внедрение).

Плюсы: Простота планирования.

Минусы: Невозможность изменений после начала этапа.

Итерационная – циклы разработки с постепенным уточнением требований.

Пример: модель Боэма (Spiral).

Гибкие методологии (Agile, 2001-настоящее время) – Scrum, Kanban.

Принципы:

Инкрементальная поставка функциональности.

Гибкость к изменениям.

Назначение моделей ЖЦ:

Управление сроками и ресурсами.

Контроль качества на каждом этапе.

Обеспечение предсказуемости процесса разработки.

Использование:

Waterfall – для проектов с четкими требованиями (например, госзаказы).

Agile – для стартапов и динамично меняющихся продуктов.

c. Стандартизация моделей жизненного цикла (ЖЦ)

Международные стандарты:

ISO/IEC 12207 – базовый стандарт для процессов ЖЦ ПО.

Определяет:

Основные процессы (разработка, эксплуатация).

Вспомогательные процессы (управление качеством, конфигурацией).

IEEE 1074 – руководство по выбору процессов ЖЦ.

CMMI (Capability Maturity Model Integration) – модель зрелости процессов.

Уровни: от хаотичных (1) до оптимизируемых (5).

Связь с методологиями:

Waterfall соответствует строгому следованию этапов ISO 12207.

Agile-процессы стандартизированы в ISO/IEC 26515 (гибкая разработка).

Применение стандартов:

Сертификация компаний (например, CMMI Level 3).

Обеспечение совместимости процессов в распределенных командах.

7. Содержание основных стандартов в области моделей жизненного цикла (ЖЦ)

a. Понятие методологий ЖЦ

Методология ЖЦ – это систематизированный подход к организации процессов разработки, внедрения и сопровождения ПО, включающий:

Определенные этапы (анализ, проектирование, кодирование, тестирование).

Правила перехода между этапами.

Артефакты (документы, модели, код).

Роли участников (аналитики, разработчики, тестировщики).

Основные категории методологий:

Каскадные (Waterfall) – строгая последовательность этапов.

Итерационные (Spiral, RUP) – циклическая разработка с уточнением требований.

Гибкие (Agile, Scrum, Kanban) – инкрементальная поставка функциональности.

Гибридные (SAFe, DevOps) – комбинация Agile и системного подхода.

b. Детальное описание основных стандартов ЖЦ с классификацией направлений работ

1. ISO/IEC 12207

Название: «Процессы жизненного цикла программного обеспечения».

Классификация процессов:

Основные процессы:

Разработка (requirements → design → implementation → testing).

Эксплуатация и сопровождение.

Вспомогательные процессы:

Управление конфигурацией (CM).

Обеспечение качества (QA).

Верификация и валидация.

Организационные процессы:

Управление проектом.

Улучшение процессов.

Применение: Универсальный стандарт для любых моделей ЖЦ (включая адаптацию под Agile).

2. IEEE 1074

Название: «Разработка процессов жизненного цикла ПО».

Ключевые элементы:

Определяет 46 процессов (например, управление рисками, планирование тестирования).

Гибкость: процессы можно комбинировать под конкретный проект.

Отличие от ISO 12207: Более детализирован, но менее популярен в промышленности.

3. CMMI (Capability Maturity Model Integration)

Название: «Модель зрелости процессов разработки».

Уровни зрелоности (1–5):

Initial (хаотичные процессы).

Managed (базовое планирование).

Defined (стандартизированные процессы).

Quantitatively Managed (управление через метрики).

Optimizing (непрерывное улучшение).

Применение: Оценка и сертификация компаний (например, для госзаказов).

4. ISO/IEC 26515 (Agile-разработка)

Название: «Процессы ЖЦ для Agile-проектов».

Особенности:

Адаптация ISO 12207 под гибкие методологии.

Акцент на:

Итеративную разработку.

Непрерывную интеграцию (CI/CD).

Работу с пользовательскими историями (User Stories).

c. Понятие и применение процедуры адаптации стандартов ЖЦ

Адаптация стандартов – это модификация процессов и артефактов стандарта под конкретный проект с учетом:

Масштаба (малый стартап vs. корпоративная система).

Домена (медицина, финансы, IoT).

Методологии (Agile, Waterfall).

Этапы адаптации:

Анализ требований:

Определение критических процессов (например, безопасность для медицинского ПО).

Выбор релевантных процессов из стандарта:

Для Agile: акцент на управлении изменениями (ISO 26515).

Для Waterfall: четкие этапы (ISO 12207).

Определение артефактов:

Документы (SRS, TDD для Waterfall).

Инкрементальные交付 (Agile-спринты).

Интеграция с существующими практиками:

Например, сочетание Scrum с CMMI Level 3.

Примеры адаптации:

Стартап: ISO 26515 + минимальная документация.

Банковский сектор: ISO 12207 + CMMI Level 4 (строгий контроль качества).

8. Основные модели ЖЦ и их области применения

a. Варианты каскадной модели

Классическая каскадная модель: этапы идут строго последовательно (анализ → проектирование → реализация → тестирование → эксплуатация).

Применение: простые, стабильные проекты с чёткими требованиями.

Каскад с обратной связью: возможен возврат на предыдущие этапы.

Применение: допускает уточнение требований.

V-модель (верификационно-валидационная): акцент на тестирование, каждому этапу разработки соответствует этап тестирования.

Применение: проекты с критически важной безопасностью (авиация, медицина).

b. Модель XP (Extreme Programming)

Особенности: частые релизы, парное программирование, непрерывное тестирование, простота дизайна, участие заказчика.

Применение: небольшие команды, нестабильные требования, быстрая обратная связь.

c. Спиральная модель (Boehm, 1988)

Итерационная модель с акцентом на управление рисками.

Каждый виток включает: определение целей → анализ рисков → разработка → оценка.

Применение: крупные, дорогостоящие, рискованные проекты.

d. Agile-модели ЖЦ

Общие принципы: инкрементальная поставка, адаптивное планирование, взаимодействие с заказчиком.

Scrum: короткие спринты, ежедневные стендапы, фиксированные роли.

Kanban: непрерывный поток задач, визуализация процесса.

Применение: быстро меняющиеся требования, активное участие заказчика.

e. Модель Гантера «фазы-функции»

Фазы (вдоль одной оси): анализ, проектирование, реализация, тестирование и т.д.

Функции (вдоль другой оси): спецификация, разработка, верификация, документация и т.д.

Особенность: позволяет адаптировать проект к любому стилю разработки, в том числе к ООП — где фазы могут перекрываться, а функции распределяются между объектами.

Применение: для наглядного представления процесса разработки в разных стилях.

9. Каноническая модель разработки ПО

a. Понятие канонической модели: область применения и значение

Каноническая модель — обобщённая модель разработки программного обеспечения, которая описывает типовые стадии и процессы, независимо от методологии (Agile, Waterfall и т.д.).

Основные стадии:

Анализ требований

Проектирование

Реализация (кодирование)

Тестирование

Внедрение (эксплуатация)

Сопровождение

Область применения: универсальна, подходит для понимания, планирования и сравнения процессов разработки.

Значение: формирует единое представление о ЖЦ ПО, независимо от конкретных моделей.

b. Технологическая сеть проектирования: содержание этапов, входные и выходные артефакты

Технологическая сеть — это детализированное представление разработки ПО в виде сети взаимосвязанных этапов, каждый из которых производит определённые артефакты (документы, модели, код и т.д.).

Этап Входные артефакты Содержание Выходные артефакты

1. Анализ требований Заявка, цели Выявление, анализ, формализация требований ТЗ (техническое задание), спецификации

2. Проектирование ТЗ Архитектура, интерфейсы, структура данных, алгоритмы Проектная документация, модели UML, прототипы

3. Реализация Проектная документация Написание программного кода, модульное тестирование Исходный код, отчёты о тестировании

4. Тестирование Код, тест-планы Интеграция, системное и приёмочное тестирование Отчёты о тестировании, исправленные баги

5. Внедрение Тестированное ПО Развёртывание, обучение пользователей Установленное ПО, руководство пользователя

6. Сопровождение Эксплуатируемое ПО, обратная связь Обновления, исправления, адаптация Новые версии, патчи, документация по изменениям

10. Управление разработкой программного проекта

a. Общие вопросы и категории управления программным проектом

Управление проектом — это планирование, организация, контроль и завершение проекта в рамках целей, сроков и бюджета.

Основные категории (области) управления:

Управление требованиями – сбор и контроль изменений требований.

Управление временем – планирование сроков, контроль графика.

Управление ресурсами – управление командой, оборудованием.

Управление стоимостью – оценка и контроль затрат.

Управление качеством – обеспечение соответствия стандартам.

Управление рисками – выявление и минимизация угроз.

Управление коммуникациями – информационные потоки в команде.

Управление конфигурациями и изменениями – отслеживание версий и модификаций.

b. Треугольник проекта (Project Management Triangle)

Три параметра:

Сроки (Time)

Стоимость (Cost)

Качество / Содержание (Scope/Quality)

📌 Суть: изменение одного параметра влияет на остальные.

Например:

– увеличить объем → ↑ время или ↑ затраты

– сократить сроки → ↓ качество или ↑ стоимость

Применение: помогает принимать сбалансированные управленческие решения при ограничениях.

c. Классификация рисков ИТ-проекта

Классы рисков:

Технические:

Непроверенные технологии

Ошибки в архитектуре, интеграции

Управленческие:

Плохое планирование

Частая смена требований

Организационные:

Недостаток ресурсов

Плохая коммуникация

Внешние:

Изменения законодательства

Влияние заказчика или поставщиков

✅ Управление рисками включает:

Идентификацию

Оценку (вероятность × ущерб)

Планирование мер

Контроль

d. Измерения и формализация проекта

Зачем нужны: для объективного анализа, контроля и улучшения.

Основные метрики:

Объём работ: LOC (строки кода), FP (функциональные точки)

Сроки: плановые и фактические

Затраты: бюджет, затраченное время

Качество: количество багов, плотность дефектов

Прогресс: процент выполнения задач (например, по диаграмме Ганта)

Формализация: ведение документации, использование инструментов управления (Jira, MS Project), диаграммы (PERT, CPM).

11. CASE-средства при управлении процессом выполнения ИТ-проекта

a. Основные фазы процесса создания ПО. Родительские и дочерние задачи

Фазы разработки ПО (по канонической модели):

Анализ требований

Проектирование

Реализация

Тестирование

Внедрение

Сопровождение

CASE-средства (Computer-Aided Software Engineering) — это ПО, которое автоматизирует управление проектом, документирование, моделирование, контроль задач и др.

Родительские задачи — это крупные этапы (например, «Разработка интерфейса»).

Дочерние задачи — более детализированные подэтапы (например, «Верстка главной страницы», «Тестирование адаптивности»).

🔧 CASE-средства помогают:

Иерархически организовать задачи

Назначать ответственных

Следить за сроками и зависимостями

b. Диаграмма Ганта и метод PERT

Диаграмма Ганта — временная шкала с задачами:

Отображает задачи, их продолжительность и взаимосвязи

Подходит для визуального контроля прогресса

Используется в большинстве проектных CASE-средств

Метод PERT (Program Evaluation and Review Technique):

Сетевое планирование с оценкой вероятностей

Используются три оценки: оптимистическая, пессимистическая, наиболее вероятная

Позволяет определить критический путь — минимально возможную продолжительность проекта

c. Известные средства управления ИТ-проектом

Средство Возможности

Jira Назначение задач, трекинг, спринты, диаграммы Ганта (через плагины), контроль времени

Microsoft Project Диаграммы Ганта, PERT, ресурсы, бюджет, критический путь

Trello Канбан-доски, контроль статуса задач, простое распределение задач

Asana Таймлайны, подзадачи, календарь, мониторинг загрузки ресурсов

ClickUp Иерархия задач, диаграммы, тайм-трекинг, распределение по команде

Redmine Отчёты, контроль задач, учёт трудозатрат, интеграция с Git

YouTrack Контроль задач, Agile-метрики, управление временем

📌 Возможности по распределению ресурсов и бюджета:

Назначение задач на конкретных участников

Учет трудозатрат (часы/дни)

Оценка стоимости по трудозатратам

Отчётность по перерасходу ресурсов

12. Начальные этапы процесса разработки ПО

a. Этап инициации проекта

Цель: обоснование необходимости проекта и принятие решения о его запуске.

Основные действия:

Формулировка бизнес-проблемы или потребности

Оценка целесообразности проекта

Назначение руководителя проекта

Определение заинтересованных сторон

Подготовка Устава проекта

b. Понятие, назначение и содержание Устава проекта

Устав проекта — официальный документ, формально запускающий проект.

Назначение:

Подтверждает существование проекта

Делегирует полномочия руководителю

Определяет ключевые параметры

Содержание:

Название и цель проекта

Обоснование (бизнес-кейс)

Основные результаты (deliverables)

Ограничения и допущения

Ключевые участники и роли

Бюджет, сроки, риски (общо)

Критерии успеха

c. Создание и содержание технического задания (ТЗ) на ИТ-проект

ТЗ (или спецификация требований) — документ, определяющий что должно быть разработано.

Содержание ТЗ:

Общие сведения (название, заказчик, разработчик)

Назначение и цели системы

Требования к функциональности

Нефункциональные требования (надежность, производительность, безопасность и т.д.)

Требования к интерфейсу

Ограничения (технологические, регламентные)

Требования к качеству и тестированию

Условия эксплуатации, сопровождения

d. Разновидности стандартов ТЗ. Последние стандарты на ТЗ к проекту создания ПО

Основные стандарты:

ГОСТ 34.602-89 — «Техническое задание на создание автоматизированной системы»

Классический стандарт в СНГ

ГОСТ 19.201-78 — старый стандарт ТЗ для программных изделий

ГОСТ Р ИСО/МЭК 25010-2014 — качество ПО (для требований к качеству)

IEEE 830 / ISO/IEC/IEEE 29148:2018 — современные международные стандарты спецификаций требований к ПО

Рекомендуются для актуальных проектов

Включают описание требований пользователей, системных требований и требований к интерфейсам

e. Последовательность начальных этапов создания ПО

Инициация проекта

Анализ бизнес-потребности

Назначение ответственных

Подготовка Устава проекта

Формализация целей и ограничений

Сбор и анализ требований

Работа с заказчиком, анализ среды

Формирование ТЗ

Детальное описание требований

Одобрение ТЗ

Подписание документа, начало проектирования

13. Процессы исполнения, завершения, мониторинга и управления проектом

a. Группа процессов исполнения проектов

Цель: реализация плана проекта и достижение результатов.

Основные процессы исполнения:

Управление исполнением работ — выполнение задач согласно плану.

Обеспечение качества — соблюдение стандартов.

Управление коммуникациями — передача информации между участниками.

Управление персоналом — распределение ролей, мотивация.

Управление закупками — работа с подрядчиками/поставщиками.

b. Метрики процессов

Метрики — количественные показатели, позволяющие отслеживать эффективность проекта.

Примеры:

SPI (Schedule Performance Index) = EV / PV

(SPI < 1 → отставание по срокам)

CPI (Cost Performance Index) = EV / AC

(CPI < 1 → перерасход бюджета)

EV (Earned Value) — заработанная стоимость

Defect density — количество дефектов на 1KLOC

Code churn — объём изменений в коде за период

c. Методы оценки степени выполнения проекта

Earned Value Management (EVM) — контроль стоимости и сроков:

Сравнение EV, PV (плановая стоимость), AC (фактические затраты)

Диаграмма Ганта — визуальное отображение прогресса по задачам

Burn-down chart (Agile) — показывает объём оставшейся работы

Вехи (milestones) — контроль по ключевым точкам проекта

d. Эвристические оценки времени выполнения проекта

Оценки времени по экспертной или приближённой оценке.

Методы:

Метод трехточечной оценки (PERT):

𝐸

=

𝑂

+

4

𝑀

+

𝑃

6

E=

6

O+4M+P

​

где O — оптимистичная, M — наиболее вероятная, P — пессимистичная оценки.

Planning Poker (Agile) — групповая экспертная оценка.

Фиксированная аналогия — на основе похожих проектов.

Delphi — анонимные экспертные оценки с консенсусом.

e. Особенности группы процессов завершения проекта

Цель: формальное завершение всех работ проекта.

Основные действия:

Проверка выполнения всех требований

Подготовка итоговой документации

Закрытие контрактов и закупок

Передача результатов заказчику

Анализ уроков проекта (lessons learned)

Освобождение ресурсов и завершение команды

f. Процессы верификации и валидации в проекте создания ПО

Верификация (verification) — «правильно ли мы реализуем требования?»

🔹 Проверка соответствия документации, кода, тестов требованиям (Code Review, Unit тесты и т.п.)

Валидация (validation) — «разрабатываем ли мы то, что нужно пользователю?»

🔹 Проверка соответствия системы ожиданиям заказчика (acceptance-тесты, демонстрации)

📌 Оба процесса важны и проводятся на всех этапах ЖЦ.

14. Информационное обеспечение ИС

a. Понятие, содержание и задачи информационного обеспечения ИС

Информационное обеспечение (ИО) — это совокупность данных, структур, методов и средств, обеспечивающих хранение, обработку и передачу информации в ИС.

Содержание:

Источники данных

Форматы, структура и объем информации

Методы кодирования, классификации, обновления данных

Задачи:

Обеспечить полноту, актуальность и достоверность информации

Организовать удобный доступ к данным

Согласовать структуру информации с задачами системы

b. Внемашинное информационное обеспечение

Это информация вне компьютера, подлежащая формализации и обработке.

Компоненты:

Классификаторы — системы кодирования и группировки данных

Документы — входные, выходные, промежуточные документы

Информационные модели — описания предметной области, структуры данных, бизнес-процессов

Нормативно-справочная информация (НСИ) — стандарты, регламенты, справочники

Назначение:

Формируют входную информационную базу

Задают правила обработки и интерпретации данных

c. Классификаторы: понятие, виды, параметры, цель использования

Классификатор — это система кодирования и группировки информации по определённым признакам.

Цель использования:

Упрощение поиска, сортировки и обработки информации

Стандартизация данных

Виды классификаторов:

Иерархические (древовидные): Коды разбиваются на уровни — например, ОКВЭД.

Фасетные (по независимым признакам): каждая характеристика кодируется отдельно.

Алфавитные, числовые, смешанные

Параметры:

Код классификации

Наименование объекта

Описание признака

Уровень классификации

d. Понятие внутримашинного информационного обеспечения

Внутримашинное ИО — это данные, хранящиеся и обрабатываемые внутри компьютера.

Сюда входят:

Базы данных

Форматы хранения (таблицы, XML, JSON и др.)

Индексы, связи, ключи

Файлы конфигурации, буферы, кэш

Место в процессе разработки ПО:

Определяется при проектировании логической и физической модели данных

Требует согласования с внемашинным ИО

Включает реализацию структуры данных, СУБД, форматов экспорта/импорта

📌 Внутримашинное ИО реализуется средствами СУБД, программной логикой, API.

15. Матрицы задач жизненного цикла и ответственности. Их роль в управлении процессом разработки ПО

✅ 1. Понятие матрицы задач жизненного цикла (ЖЦ ПО)

Матрица задач ЖЦ ПО — это таблица, где перечислены:

Этапы/фазы жизненного цикла ПО (анализ, проектирование, реализация, тестирование и т.д.)

Работы (процессы), выполняемые на каждом этапе

Ответственные роли (разработчики, аналитики, тестировщики и т.п.)

📌 Назначение:

Стандартизирует процессы

Упрощает контроль исполнения

Показывает, кто и за что отвечает на каждом этапе

✅ 2. Матрица ответственности (например, RACI)

RACI — популярная модель распределения ответственности:

Обозначение Расшифровка Роль

R Responsible Выполняет задачу

A Accountable Несёт ответственность (утверждает)

C Consulted Консультирует

I Informed Уведомляется

📌 Пример — задача: "Разработка интерфейса":

UI-дизайнер: R

Руководитель проекта: A

Бизнес-аналитик: C

Заказчик: I

✅ 3. Роль матриц в управлении проектом

Чётко определяют ответственность и взаимодействие между участниками

Снижают риск пропущенных задач

Обеспечивают трассируемость требований на всех этапах ЖЦ

Облегчают аудит и контроль качества

Улучшают планирование ресурсов

✅ 4. Где применяются

В рамках стандартов разработки ПО (например, ГОСТ 34, ISO/IEC 12207)

В методологиях управления проектами (PMBOK, PRINCE2)

В Agile и Scrum: используется упрощённо, в рамках ролей команды (PO, Dev, QA)

16. Требования к ПО в процессе разработки

✅ a. Понятие требований к ПО, классификация

Требования к ПО — это описание того, что должно делать программное обеспечение, при каких условиях и с какими ограничениями.

Классификация требований:

Функциональные требования:

Что должно делать ПО: функции, реакции на действия пользователя, поведение в сценариях.

Пример: «ПО должно позволять пользователю сохранять файл в формате PDF».

Нефункциональные требования:

Как должно работать ПО: производительность, безопасность, надёжность, масштабируемость, интерфейс.

Пример: «Ответ на запрос должен быть не позже чем через 2 секунды».

Бизнес-требования:

Цели проекта с точки зрения заказчика.

Системные требования:

Аппаратное и программное окружение, зависимости, ограничения.

Пользовательские требования:

Требования, сформулированные с точки зрения конечного пользователя (use cases, user stories).

✅ b. Процесс разработки требований (этапы)

Сбор требований:

Интервью, опросы, анализ документов, наблюдение.

Анализ и интерпретация:

Уточнение, устранение противоречий и дублирования.

Документирование:

Создание спецификаций (SRS), user stories, моделей.

Проверка (верификация):

Проверка на полноту, непротиворечивость, выполнимость.

Утверждение:

Подписание документа заказчиком и командой.

Управление изменениями:

Ведение версии требований при изменении проекта.

✅ c. Требования и спецификации

Требования — это что нужно системе (на уровне идеи и задачи).

Спецификация требований (SRS) — это формализованный документ, в котором требования представлены чётко, полно и однозначно.

Особенности спецификации:

Стандарты: IEEE 830 / ISO/IEC/IEEE 29148:2018

Форматы: текстовые описания, таблицы, схемы (UML, BPMN), user stories

Может включать:

Общие сведения

Сценарии использования

Диаграммы состояний, классов, потоков данных

Требования к качеству

📌 Хорошая спецификация должна быть:

Полной

Проверяемой

Однозначной

Прослеживаемой

17. Архитектура программного обеспечения

✅ a. Временная эволюция понятия архитектуры ПО

Ранний этап (1970–80-е):

Архитектура рассматривалась как структура модулей и их взаимодействия (структурный подход).

Главная цель — управление сложностью программных систем.

1990-е:

Введение понятия архитектурных стилей (client-server, layered и др.)

Акцент на качественные атрибуты: надёжность, расширяемость, производительность.

2000-е и далее:

Появление SOA (сервисно-ориентированная архитектура), позже микросервисов

Связь архитектуры ПО с архитектурой предприятия

Внедрение архитектурных фреймворков (TOGAF, Zachman, 4+1)

✅ b. Об архитектуре информационной системы (ИС)

Архитектура ИС — структура и взаимодействие компонентов системы, обеспечивающих обработку, хранение и передачу информации.

Состав:

Архитектура ПО (приложений и сервисов)

Архитектура данных (модели хранения, взаимодействия)

Техническая (инфраструктурная) архитектура

Безопасность, интеграция, интерфейсы

✅ c. 4 типа архитектуры ПО и принципы их работы

Клиент-серверная архитектура

Сервер предоставляет данные и сервисы, клиент — интерфейс.

Пример: веб-приложение.

Слоистая (layered) архитектура

Логическое разделение на уровни: представление, бизнес-логика, данные.

Принцип: каждый слой взаимодействует только с соседним.

Микросервисная архитектура

Приложение делится на мелкие независимые сервисы.

Принцип: изоляция, независимое развертывание, API-интеграция.

Событийно-ориентированная архитектура (EDA)

Компоненты реагируют на события, публикуемые в системе.

Применяется в распределённых, real-time системах.

📌 Также популярны: монолит, SOA, потоковая, архитектура REST.

✅ d. Архитектура цифрового предприятия и модель Захмана

Архитектура цифрового предприятия:

Интеграция ИТ-систем, бизнес-процессов и данных.

Основывается на принципах цифровизации: гибкость, масштабируемость, автоматизация.

Модель Захмана (Zachman Framework):

Архитектурный фреймворк предприятия.

6 вопросов (что, как, где, кто, когда, почему) × 6 ролей (планировщик, владелец, дизайнер, строитель, подрядчик, пользователь).

📌 Связь: архитектура ПО реализует ИТ-аспекты бизнес-архитектуры из модели Захмана.

✅ e. Задачи архитектора ПО и уровни архитектуры

Задачи архитектора:

Проектировать архитектуру системы

Обеспечивать выполнение нефункциональных требований (надежность, масштабируемость и т.д.)

Выбирать технологии, интеграционные решения

Документировать архитектуру

Консультировать команду

Архитектура верхнего уровня (high-level):

Общая структура компонентов

Точки взаимодействия (API, базы данных, шины сообщений)

Архитектура нижнего уровня (low-level):

Детализация компонентов, классов, взаимодействий

Проектирование модулей и интерфейсов

Результаты работы архитектора:

Архитектурное описание (AD) — диаграммы, схемы, спецификации

Архитектурные решения (ADR)

Нефункциональные требования

Используются при:

Проектировании и разработке

Тестировании (особенно нагрузочном)

Поддержке и масштабировании

18 Стоимость разработки ПО

a. Основные проблемы и задачи оценки стоимости разработки ПО

i. Обоснование необходимости разработки (Технико-экономическое обоснование — ТЭО):

Для заказчика: понять целесообразность проекта, ожидаемые выгоды, риски и затраты.

Для разработчика: оценить ресурсы, сроки, экономическую эффективность работы.

ТЭО — документ, включающий анализ затрат и выгод, основание для принятия решения о запуске проекта.

ii. Точность и этапность оценки:

На ранних этапах — грубая оценка (с точностью ±30-50%) для принятия решения.

По мере детализации проекта — уточнение оценки (±10-20%).

Этапность необходима для адаптации бюджета и ресурсов.

b. Подходы к оценке стоимости ПО

COCOMO (Constructive Cost Model):

Модель, основанная на размере кода (в строках кода — LOC).

Классическая COCOMO имеет три уровня: базовый, промежуточный и детальный.

Формула: трудоёмкость = a \* (размер)^b \* коэффициенты коррекции.

COCOMO-II:

Современная версия, учитывающая современные технологии, методы разработки и архитектуру.

Вводит новые факторы (например, масштаб проекта, опыт команды, использование инструментов).

Метод функциональных точек (Function Points, FP):

Оценивает размер системы на основе функций, доступных пользователю (входные данные, выходные данные, запросы, файлы).

Не зависит от языка программирования и объёма кода.

Позволяет оценивать трудоёмкость по количеству функциональных точек, умноженных на среднюю трудоёмкость.

c. Статистические закономерности в оценке трудоёмкости

Закон Парето: 20% усилий дают 80% результата, важные задачи концентрируют большую часть затрат.

Трудоёмкость пропорциональна экспоненте размера проекта (увеличение размера резко повышает трудозатраты).

Существуют закономерности зависимости затрат от квалификации команды, сложности, технологии.

Типичная ошибка — недооценка влияния коммуникаций и управления в больших проектах.

19. Риски проекта разработки ПО и метрики качества ПО

a. Риски проекта разработки ПО

Основные категории рисков:

Технические риски

Недостаточная техническая компетентность, ошибки проектирования, сложность технологии.

Организационные риски

Нехватка ресурсов, плохое управление, конфликты в команде.

Коммерческие риски

Изменение требований, бюджетные ограничения, задержки поставок.

Внешние риски

Изменение законодательства, экономическая нестабильность, форс-мажор.

Риски качества

Недостаточное тестирование, баги, проблемы с производительностью.

b. Меры предупреждения и отработки рисков

Идентификация рисков (раннее выявление)

Оценка рисков (вероятность и влияние)

Планирование реакций:

Избежание риска (изменение плана)

Снижение риска (меры контроля)

Передача риска (аутсорсинг, страхование)

Принятие риска (готовность к последствиям)

Мониторинг и контроль рисков на протяжении проекта

Документирование уроков и корректировка планов

c. Параметры качества ПО и связь с моделью CCMI

Основные параметры качества ПО (на основе модели ISO/IEC 25010):

Функциональность — соответствие требованиям

Надежность — стабильность работы, отказоустойчивость

Используемость (удобство) — удобство для пользователя

Производительность — скорость и эффективность

Поддерживаемость — простота внесения изменений

Безопасность — защита от несанкционированного доступа

Совместимость — работа с другими системами

Модель CCMI (Capability and Maturity Model Integration):

Фреймворк оценки зрелости процессов разработки и качества

Связывает параметры качества с процессными практиками

Помогает систематически улучшать качество через зрелость процессов

Качество ПО напрямую зависит от зрелости процессов CCMI на уровне организации