Оглавление

[1 Стандарты качества ISO в применении к программному обеспечению 2](#_Toc154654584)

[2 Спецификация требований в программной инженерии. Основные разделы и их содержание 2](#_Toc154654585)

[3 Процесс анализа требований в программной инженерии. Требования и их свойства 2](#_Toc154654586)

[4 Нотации и языки моделирования предметной области в программной инженерии 2](#_Toc154654587)

[5 Современные проблемы и тенденции в области разработки и использования информационных технологий и программной инженерии 2](#_Toc154654588)

[6 Расширенный анализ требований в программной инженерии. Моделирование и прототипирование. Спецификация системных требований 2](#_Toc154654589)

[7 Концептуальное моделирование в программной инженерии 2](#_Toc154654590)

[8 Модели жизненного цикла, существующие и перспективные модели жизненного цикла программного обеспечения, их достоинства и недостатки 2](#_Toc154654591)

[9 Парадигмы программирования при проектировании программного обеспечения в программной инженерии 2](#_Toc154654592)

[10 Модельно-ориентированный подход к проектированию программного обеспечения в программной инженерии 2](#_Toc154654593)

[11 Организация технологического процесса промышленного производства программного обеспечения и его коллективной разработки 2](#_Toc154654594)

[12 Технологии управления проектами в промышленном производстве программного обеспечения и программной инженерии 2](#_Toc154654595)

[13 Технологии управления версионированием исходных кодов в промышленном производстве программного обеспечения и программной инженерии 2](#_Toc154654596)

[14 Технологии общего и системного программного обеспечения в программной инженерии 2](#_Toc154654597)

[15 Web-технологии в промышленном производстве программного обеспечения и программной инженерии 2](#_Toc154654598)

[16 Технологии организации хранения данных в промышленном производстве программного обеспечения и программной инженерии 2](#_Toc154654599)

[17 Технологии разработки и архитектура специального программного обеспечения в программной инженерии 3](#_Toc154654600)

[18 Концепции автоматизации процессов производства программного обеспечения на этапах его разработки и внедрения 3](#_Toc154654601)

[19 Классификация информационных систем в программной инженерии 3](#_Toc154654602)

[20 Управление конфигурациями программного обеспечения в программной инженерии 3](#_Toc154654603)

[21 Автоматизация компиляции и компоновки программного обеспечения в программной инженерии 3](#_Toc154654604)

[22 Технологии непрерывной интеграции комплексов программного обеспечения в программной инженерии 3](#_Toc154654605)

[23 Непрерывная поставка и развертывание программного обеспечения в программной инженерии 3](#_Toc154654606)

[24 Управление качеством в программной инженерии. Показатели качества программного обеспечения 3](#_Toc154654607)

[25 Виды испытаний программного обеспечения в соответствии с международными и отечественными стандартами и рекомендациями в программной инженерии 3](#_Toc154654608)

[26 Тестирование программного обеспечения в программной инженерии. Цели и виды тестирования. Основные фазы тестирования 3](#_Toc154654609)

[27 Оптимизация и анализ исходных кодов программного обеспечения в программной инженерии 3](#_Toc154654610)

[28 Автоматизация тестирования программного обеспечения в программной инженерии 3](#_Toc154654611)

[29 Существующие модели качества программного обеспечения в программной инженерии, их достоинства и недостатки 3](#_Toc154654612)

[30 Основные понятия, определения и стандарты программной инженерии 3](#_Toc154654613)

[31 Жизненный цикл разработки программного обеспечения и его основные аспекты 3](#_Toc154654614)

[32 Проблема выбора модели жизненного цикла программного обеспечения и возможные пути ее решения 3](#_Toc154654615)

[33 Тестирование программного обеспечения в программной инженерии. Техники тестирования и верификация программ 4](#_Toc154654616)

[34 Интеллектуальные информационные технологии в программной инженерии 4](#_Toc154654617)

[35 Отечественные и зарубежные стандарты документирования программного обеспечения. Типы программной документации 4](#_Toc154654618)

[36 Типы и назначение документации программного обеспечения в соответствии с Единой системой программной документации. 4](#_Toc154654619)

[37 Комплекс стандартов на автоматизированные системы в программной инженерии 4](#_Toc154654620)

[38 Технологии формирования и ведения баз знаний при разработке программного обеспечения в программной инженерии 4](#_Toc154654621)

[39 Классификация операционных систем в программной инженерии. Операционные системы на базе GNU/Linux. Операционные системы специального назначения в промышленном производстве программного обеспечения 4](#_Toc154654622)

[40 Технологии виртуализации и контейнеризации в операционных системах на этапах разработки и эксплуатации программного обеспечения 4](#_Toc154654623)

# Стандарты качества ISO в применении к программному обеспечению

**Стандарты качества ISO в применении к программному обеспечению**

**1. Общее понимание ISO и его важность в программной инженерии:**

* **ISO (Международная организация по стандартизации)** представляет собой независимую организацию, разрабатывающую и публикующую международные стандарты в различных областях, включая программное обеспечение и ИТ.
* **Значение для программной инженерии:** Стандарты ISO помогают обеспечить качество, безопасность, эффективность и взаимозаменяемость продуктов и услуг.

**2. Основные стандарты ISO для программного обеспечения:**

* **ISO/IEC 90003:** Предоставляет руководство по применению ISO 9001 в процессах разработки, поставки и обслуживания программного обеспечения.
* **ISO/IEC 25010 (SQuaRE - System and Software Quality Requirements and Evaluation):** Описывает модель качества программного обеспечения, включая такие аспекты, как функциональность, надежность, удобство использования, эффективность, обслуживаемость и переносимость.
* **ISO/IEC 27001:** Стандарт по информационной безопасности, важный для разработки и поддержки безопасного программного обеспечения.

**3. Применение стандартов ISO в разработке ПО:**

* **Планирование качества:** Определение стандартов и процедур, соответствующих ISO, которые будут применяться в проекте.
* **Разработка и проектирование:** Применение стандартов ISO при проектировании архитектуры ПО и выборе технологий.
* **Тестирование и проверка качества:** Оценка соответствия продукта стандартам ISO через комплексное тестирование и аудит.
* **Обслуживание и поддержка:** Соблюдение стандартов ISO в процессах обновления, исправления ошибок и техподдержки.

**4. Преимущества соблюдения стандартов ISO:**

* **Улучшение качества продукта:** Стандарты помогают улучшить надежность, производительность и безопасность ПО.
* **Увеличение доверия клиентов и рыночной конкурентоспособности:** Сертификация ISO может служить доказательством качества для клиентов и партнеров.
* **Снижение рисков:** Стандарты помогают минимизировать риски в процессе разработки и эксплуатации ПО.

**5. Вызовы применения стандартов ISO:**

* **Затраты на внедрение:** Разработка и поддержка процессов, соответствующих стандартам ISO, требуют времени и ресурсов.
* **Необходимость обучения и адаптации:** Командам разработчиков может потребоваться обучение для понимания и правильного применения стандартов.
* **Гибкость и адаптивность:** Необходимо гармонично сочетать стандарты ISO с гибкими методологиями разработки, чтобы избежать излишней бюрократии.

**6. Перспективы и развитие:**

* Стандарты ISO постоянно обновляются для соответствия меняющимся технологиям и требованиям рынка.
* Растет важность интеграции стандартов качества ISO с современными подходами в разработке программного обеспечения, такими как DevOps и Agile.

# 2 Спецификация требований в программной инженерии. Основные разделы и их содержание

# 3 Процесс анализа требований в программной инженерии. Требования и их свойства

**Процесс анализа требований в программной инженерии**

**1. Определение анализа требований:**

* **Анализ требований** — это процесс определения пользовательских ожиданий для новой или изменяемой продукта. Этот процесс включает в себя выявление, оценку и документирование потребностей и ограничений проекта.

**2. Этапы процесса анализа требований:**

* **Сбор требований:** Взаимодействие с заинтересованными сторонами для получения ясного понимания их потребностей и ожиданий.
* **Классификация требований:** Разделение собранных требований на категории, такие как функциональные, нефункциональные, технические и бизнес-требования.
* **Анализ и уточнение требований:** Проверка требований на полноту, непротиворечивость, реализуемость и необходимость.
* **Приоритизация требований:** Определение приоритетов требований на основе важности и влияния на проект.
* **Документирование и утверждение требований:** Формализация требований в документе спецификации требований и получение одобрения от заинтересованных сторон.

**3. Требования и их свойства:**

* **Функциональные требования:** Описывают конкретные функции и поведение системы (например, возможности, операции, логику работы).
* **Нефункциональные требования:** Определяют атрибуты системы, такие как производительность, безопасность, масштабируемость и удобство использования.
* **Ограничения:** Ограничения или ограничивающие факторы, которые влияют на разработку (например, законодательные нормы, сроки, бюджет).

**4. Свойства хорошо сформулированных требований:**

* **Конкретность:** Требования должны быть четко и ясно сформулированы.
* **Измеримость:** Требования должны быть проверяемыми, то есть должна существовать возможность проверить их выполнение.
* **Реализуемость:** Требования должны быть выполнимыми с технологической и финансовой точек зрения.
* **Релевантность:** Требования должны соответствовать целям и потребностям бизнеса или пользователя.
* **Консистентность:** Отсутствие противоречий в требованиях.

**5. Инструменты и методики анализа требований:**

* **Использование моделей и диаграмм:** Для визуализации и лучшего понимания требований (например, UML).
* **Прототипирование:** Разработка прототипов для демонстрации и проверки ключевых аспектов системы.
* **Сессии мозгового штурма и интервью:** Для генерации и уточнения идей и требований.
* **Использование специализированного ПО:** Такие инструменты, как JIRA, Trello или Rational RequisitePro, могут использоваться для управления требованиями.

**6. Заключение:**

* Анализ требований — критически важная часть процесса разработки ПО. Он определяет фундамент проекта и влияет на все последующие этапы разработки. Правильное понимание и управление требованиями могут значительно сократить риски, связанные с изменениями требований, и обеспечить успешную реализацию проекта.

# 4 Нотации и языки моделирования предметной области в программной инженерии

# 5 Современные проблемы и тенденции в области разработки и использования информационных технологий и программной инженерии

В области разработки и использования информационных технологий и программной инженерии существует ряд современных проблем и тенденций, которые активно обсуждаются и исследуются в настоящее время:

1. **Безопасность и конфиденциальность данных**: По мере того как всё больше данных переходят в цифровой формат и хранятся в облаке, вопросы безопасности и конфиденциальности становятся всё более актуальными. Разработчики сталкиваются с задачей создания надёжных и безопасных систем, способных противостоять кибератакам и утечкам данных.
2. **Искусственный интеллект и машинное обучение**: Интеграция ИИ и машинного обучения в различные аспекты программной инженерии меняет подход к разработке ПО. Это создаёт новые возможности, но и предъявляет требования к навыкам разработчиков, а также порождает этические и нормативные вопросы.
3. **DevOps и автоматизация процессов**: Подходы DevOps, направленные на интеграцию разработки и операционной деятельности, становятся стандартом. Вместе с этим растёт спрос на автоматизацию процессов разработки, тестирования и развёртывания.
4. **Облачные технологии и микросервисная архитектура**: Облачные платформы и микросервисы значительно изменили подход к разработке и развёртыванию приложений. Это позволяет повысить масштабируемость и гибкость систем, но также увеличивает сложность управления и мониторинга.
5. **Контейнеризация и оркестрация**: Использование контейнеров (например, Docker) и систем оркестрации (например, Kubernetes) стало ключевой практикой для обеспечения эффективного развертывания и масштабирования приложений.
6. **Устойчивое развитие и экологическая ответственность**: С ростом осведомленности о проблемах изменения климата увеличивается давление на ИТ-индустрию с целью сокращения углеродного следа, использования энергоэффективных технологий и устойчивого производства.
7. **Удалённая работа и распределённые команды**: Пандемия COVID-19 ускорила переход к удалённой работе, что стало нормой для многих разработчиков. Это создаёт новые вызовы в управлении проектами и коммуникациях внутри команд.
8. **Обновление навыков и обучение**: Постоянно меняющийся технологический ландшафт требует от профессионалов постоянного обучения и обновления навыков для поддержания актуальности их знаний.
9. **Регулирование и соответствие стандартам**: Ужесточение законодательства в области цифровой безопасности, конфиденциальности данных (например, GDPR) требует от компаний соответствия новым нормативам.
10. **Этические и социальные аспекты**: Растущее внимание к этическим вопросам, связанным с разработкой и использованием технологий, таких как предвзятость в ИИ, влияние технологий на общество и работу.

Эти тенденции и проблемы непрерывно развиваются и влияют на то, как компании и индивидуальные разработчики подходят к проектированию, разработке и поддержке программного обеспечения.

# 6 Расширенный анализ требований в программной инженерии. Моделирование и прототипирование. Спецификация системных требований

# 7 Концептуальное моделирование в программной инженерии

Концептуальное моделирование в программной инженерии - это процесс создания абстрактных моделей систем или проектов, которые отражают ключевые концепции и взаимоотношения в рамках данного контекста, не углубляясь в технические детали. Этот подход используется для упрощения сложных систем и обеспечения общего понимания их структуры и функционирования среди всех участников проекта. Вот основные аспекты и преимущества концептуального моделирования:

1. **Формализация требований и целей проекта**: Концептуальные модели помогают определить основные требования и цели проекта, позволяя всем участникам иметь четкое понимание задач и ожиданий.
2. **Унификация понимания**: Эти модели служат средством коммуникации между всеми заинтересованными сторонами, включая разработчиков, менеджеров проектов, клиентов и пользователей. Они обеспечивают общий язык для обсуждения и понимания проекта.
3. **Упрощение сложных систем**: Концептуальное моделирование позволяет абстрагироваться от технических деталей и сосредоточиться на основных принципах и структурах системы.
4. **Использование стандартных нотаций**: Часто для создания концептуальных моделей используются стандартные нотации, такие как UML (Unified Modeling Language), что способствует лучшему взаимопониманию и стандартизации документации.
5. **Раннее выявление проблем и несоответствий**: Построение концептуальной модели до начала разработки помогает выявить потенциальные проблемы, несоответствия и противоречия в требованиях.
6. **Планирование и прогнозирование**: Эти модели помогают в планировании этапов разработки и могут использоваться для прогнозирования ресурсов, времени и стоимости проекта.
7. **Основа для дальнейшего проектирования**: Концептуальная модель часто служит основой для создания более детальных технических моделей и архитектур.
8. **Гибкость и масштабируемость**: Модели легко адаптируются и масштабируются при изменении требований или условий проекта.

В целом, концептуальное моделирование является ключевым этапом в процессе программной инженерии, позволяющим улучшить качество и эффективность разработки за счет четкого понимания и формализации требований и структуры проекта.

# 8 Модели жизненного цикла, существующие и перспективные модели жизненного цикла программного обеспечения, их достоинства и недостатки

# 9 Парадигмы программирования при проектировании программного обеспечения в программной инженерии

Парадигмы программирования играют ключевую роль в проектировании программного обеспечения в области программной инженерии. Парадигма программирования представляет собой набор концепций и практик, которые определяют подход к программированию и структурированию кода. Вот некоторые из наиболее распространенных парадигм:

1. **Императивное программирование**: Эта парадигма фокусируется на описании шагов, которые компьютер должен выполнить для достижения определенной цели. Она включает в себя процедурное программирование, где код организуется в функции, и объектно-ориентированное программирование, где код организован в объекты, объединяющие данные и функции, работающие с этими данными.
   * **Процедурное программирование**: Структурирует программы как набор последовательных процедур или подпрограмм.
   * **Объектно-ориентированное программирование (ООП)**: Сосредотачивается на создании объектов, которые содержат и данные, и методы для работы с этими данными.
2. **Декларативное программирование**: Вместо шагов описывает желаемый результат. Эта парадигма делит код на более высокоуровневые конструкции, что часто упрощает понимание и поддержку кода.
   * **Функциональное программирование**: Использует функции без побочных эффектов и обрабатывает данные как неизменяемые. Это упрощает тестирование и отладку.
   * **Логическое программирование**: Программы состоят из набора фактов и правил, на основе которых система делает выводы (например, Prolog).
3. **Структурное программирование**: Основано на идее использования подпрограмм, циклов и условных операторов. Эта парадигма противопоставляется прыжкам в коде (например, использование GOTO в старых языках), что улучшает читаемость и поддерживаемость кода.
4. **Реактивное программирование**: Ориентировано на потоки данных и распространение изменений. Это означает, что программы легко адаптируются к изменениям исходных данных или пользовательского ввода в реальном времени.
5. **Событийно-ориентированное программирование**: Особенно популярно в разработке пользовательских интерфейсов, где программа реагирует на различные события (например, клики мыши, нажатия клавиш).

Выбор парадигмы программирования зависит от множества факторов, включая конкретные требования проекта, предпочтения команды, доступные инструменты и языки программирования, а также от природы задачи, которую нужно решить. Нередко разработчики комбинируют элементы различных парадигм, чтобы максимально эффективно решить поставленные перед ними задачи.

# 10 Модельно-ориентированный подход к проектированию программного обеспечения в программной инженерии

# 11 Организация технологического процесса промышленного производства программного обеспечения и его коллективной разработки

Организация технологического процесса промышленного производства программного обеспечения и его коллективной разработки – это сложная задача, требующая применения различных методологий, инструментов и лучших практиик. Вот основные аспекты, которые следует учитывать:

**1. Методологии разработки ПО**

* **Гибкие методологии (Agile):** Scrum, Kanban и другие подходы Agile подчеркивают итеративное развитие, гибкость, непрерывное улучшение и активное вовлечение заказчика в процесс.
* **Водопадная модель (Waterfall):** Традиционный, последовательный подход, который хорошо работает для проектов с четко определенными требованиями и ограничениями.
* **Гибридные модели:** Комбинируют элементы гибких и традиционных подходов для достижения баланса между гибкостью и структурой.

**2. Планирование и оценка проекта**

* **Определение требований:** Четкое понимание того, что должно быть достигнуто, является ключевым фактором успешного проекта.
* **Расчет времени и ресурсов:** Использование методик, таких как покер планирования или метод Дельфи, для оценки ресурсов, времени и бюджета.

**3. Коллективная работа и командная динамика**

* **Коммуникация:** Эффективное общение и регулярные совещания (ежедневные стендапы в Agile, например) для обеспечения синхронизации команды.
* **Инструменты совместной работы:** Использование систем управления проектами (например, JIRA), инструментов для контроля версий (Git) и платформ для совместной работы (Slack, Microsoft Teams).

**4. Технические аспекты**

* **Контроль версий:** Использование систем управления версиями, таких как Git, для отслеживания изменений в коде и упрощения коллективной работы.
* **Непрерывная интеграция и доставка (CI/CD):** Автоматизация процессов сборки, тестирования и развертывания для ускорения циклов разработки и повышения качества ПО.

**5. Качество и тестирование ПО**

* **Автоматизация тестирования:** Применение инструментов автоматизации тестирования для улучшения качества и сокращения времени на тестирование.
* **Контроль качества:** Проведение код-ревью, использование статического анализа кода и интеграция тестирования безопасности в процесс разработки.

**6. Управление рисками и изменениями**

* **Анализ и управление рисками:** Идентификация потенциальных рисков и разработка планов по их минимизации.
* **Управление изменениями:** Поддержание гибкости для внесения изменений в проект, одновременно контролируя их влияние.

**7. Соблюдение норм и стандартов**

* **Соблюдение законодательства и стандартов:** Учет требований GDPR, HIPAA и других регуляторных стандартов в процессе разработки.

**8. Постоянное обучение и адаптация**

* **Обучение и развитие:** Постоянное обучение и адаптация к новым технологиям и лучшим практикам в быстро меняющейся среде IT.

Успешная организация технологического процесса в промышленном производстве ПО требует комплексного подхода, охватывающего все эти аспекты, а также гибкости для адаптации к конкретным условиям и требованиям каждого отдельного проекта.

# 12 Технологии управления проектами в промышленном производстве программного обеспечения и программной инженерии

# 13 Технологии управления версионированием исходных кодов в промышленном производстве программного обеспечения и программной инженерии

Технологии управления версионированием исходных кодов играют критически важную роль в промышленном производстве программного обеспечения и программной инженерии. Они обеспечивают эффективное управление изменениями, которые происходят в процессе разработки программного обеспечения. Ниже представлены ключевые аспекты и технологии, связанные с управлением версиями:

1. **Основы управления версиями**: Системы управления версиями (Version Control Systems, VCS) позволяют разработчикам отслеживать и управлять изменениями в исходном коде программного обеспечения. Это включает в себя возможность возвращаться к предыдущим версиям кода, сравнивать изменения между версиями и управлять различными ветками разработки.
2. **Централизованные против распределенных систем**: Ранее преобладали централизованные системы управления версиями, такие как CVS и Subversion (SVN), где все изменения хранились на центральном сервере. Современные тенденции сместились в сторону распределенных систем управления версиями, таких как Git и Mercurial, которые позволяют каждому разработчику иметь локальную копию всего репозитория, включая всю историю изменений.
3. **Git и GitHub**: Git является одной из самых популярных систем управления версиями на сегодняшний день. GitHub, GitLab и Bitbucket предлагают облачные сервисы для хостинга репозиториев Git, а также предоставляют дополнительные инструменты для совместной работы, отслеживания задач, код-ревью и интеграции с другими инструментами.
4. **Ветвление и слияние**: Ветвление (branching) и слияние (merging) являются ключевыми понятиями в управлении версиями. Они позволяют разработчикам работать над различными задачами или экспериментами параллельно, не мешая основной (продуктивной) линии разработки, а затем сливать свои изменения обратно.
5. **Управление релизами и тегирование**: Системы управления версиями позволяют отмечать определенные точки в истории кода как значимые, например, релизы продукта. Теги используются для обозначения таких значимых моментов, упрощая навигацию по версиям продукта.
6. **Интеграция с CI/CD**: Системы управления версиями тесно интегрированы с инструментами непрерывной интеграции и непрерывной доставки (CI/CD), такими как Jenkins, Travis CI, GitLab CI и другими. Это позволяет автоматизировать процессы тестирования и развертывания программного обеспечения при каждом изменении кода.
7. **Безопасность и контроль доступа**: Важным аспектом управления версиями является контроль доступа к репозиторию и исходному коду. Системы могут предоставлять различные уровни доступа для разных членов команды, а также обеспечивать безопасность данных через шифрование и безопасные протоколы передачи данных.
8. **Отслеживание проблем и интеграция с системами управления задачами**: Современные системы управления версиями часто интегрируются с системами отслеживания проблем и управления задачами, такими как Jira, Trello и другими, позволяя связывать изменения в коде с конкретными задачами или проблемами.

В целом, технологии управления версионированием исходных кодов являются фундаментальным инструментом в процессе разработки программного обеспечения, обеспечивая эффективную координацию работы команды, управление изменениями, отслеживание истории разработки и поддержание высоких стандартов качества.

# 14 Технологии общего и системного программного обеспечения в программной инженерии

# 15 Web-технологии в промышленном производстве программного обеспечения и программной инженерии

Ваш запрос охватывает важную и многогранную тему: роль и применение веб-технологий в промышленном производстве программного обеспечения и программной инженерии. Давайте разберёмся с этим в нескольких ключевых аспектах.

**1. Разработка и Развертывание ПО**

**Инструменты и Платформы:** Современные веб-технологии предоставляют мощные инструменты и платформы для разработки программного обеспечения. Это включает в себя интегрированные среды разработки (IDE), системы управления версиями (например, GitHub), и системы непрерывной интеграции/непрерывного развертывания (CI/CD), такие как Jenkins, Travis CI и GitHub Actions.

**Облачные Решения:** Облачные технологии (например, AWS, Azure, Google Cloud) позволяют разработчикам быстро развертывать и масштабировать приложения, а также предоставляют доступ к расширенным инструментам аналитики и управления данными.

**2. Сотрудничество и Управление Проектами**

**Инструменты для Совместной Работы:** Веб-технологии упрощают сотрудничество в командах разработчиков. Платформы типа Slack, Microsoft Teams и Jira облегчают общение, планирование задач и отслеживание прогресса проекта.

**Документация и Хранение Знаний:** Системы управления документацией (например, Confluence) и базы знаний в сети позволяют удобно хранить и делиться информацией внутри команды.

**3. Тестирование и Качество ПО**

**Автоматизация Тестирования:** Веб-технологии обеспечивают инструменты для автоматизации тестирования как фронтенд, так и бэкенд компонентов. Фреймворки, такие как Selenium, Puppeteer и Cypress, позволяют автоматизировать тестирование веб-интерфейсов.

**Мониторинг и Отладка:** Различные онлайн-инструменты и сервисы (например, Sentry, New Relic) используются для мониторинга производительности приложений и отладки в реальном времени.

**4. Безопасность и Соответствие Нормативам**

**Безопасность Веб-Приложений:** Веб-технологии включают инструменты и практики для обеспечения безопасности веб-приложений, включая шифрование, аутентификацию, защиту от XSS и CSRF атак.

**Соответствие Нормативам:** Инструменты для управления соответствием помогают следить за тем, чтобы программные продукты соответствовали законодательным и отраслевым стандартам (например, GDPR, HIPAA).

**5. Обучение и Развитие**

**Онлайн-Обучение:** Веб-технологии предоставляют доступ к обширным ресурсам для обучения и самосовершенствования разработчиков, включая онлайн-курсы, вебинары, туториалы.

**Сообщества и Форумы:** Платформы такие как Stack Overflow, GitHub и специализированные форумы предоставляют место для обмена знаниями, обсуждения проблем и поиска решений.

**Заключение**

Веб-технологии играют центральную роль в современной программной инженерии, предоставляя инструменты и платформы для разработки, тестирования, развертывания и поддержки программного обеспечения. Они также способствуют более эффективному взаимодействию внутри команд, обеспечивают безопасность приложений и способствуют обучению и профессиональному развитию инженеров.

# 16 Технологии организации хранения данных в промышленном производстве программного обеспечения и программной инженерии

# 17 Технологии разработки и архитектура специального программного обеспечения в программной инженерии

Тема "Технологии разработки и архитектура специального программного обеспечения в программной инженерии" включает в себя несколько ключевых аспектов. Давайте рассмотрим их более подробно:

**1. Технологии разработки программного обеспечения**

Разработка программного обеспечения включает в себя множество различных технологий и подходов. Среди наиболее значимых:

* **Языки программирования:** Различные языки (например, Java, Python, C#, JavaScript) используются для разных видов проектов в зависимости от их требований к производительности, масштабируемости и легкости поддержки.
* **Фреймворки и библиотеки:** Фреймворки (например, React для веб-разработки, Spring для Java) и библиотеки предоставляют готовые решения и шаблоны для распространенных задач, ускоряя процесс разработки.
* **Среды разработки и инструменты:** Интегрированные среды разработки (IDEs), такие как Visual Studio, IntelliJ IDEA, и системы контроля версий (например, Git) являются неотъемлемой частью процесса разработки.

**2. Архитектура программного обеспечения**

Архитектура программного обеспечения – это структурное проектирование программных систем. Важные аспекты:

* **Шаблоны проектирования:** Понимание и применение шаблонов проектирования (например, MVC, Singleton, Factory) помогает в создании гибкой, масштабируемой и легко поддерживаемой архитектуры.
* **Микросервисы и монолитные архитектуры:** Выбор между микросервисной и монолитной архитектурой зависит от масштаба и требований проекта.
* **Высокопроизводительные и масштабируемые системы:** Важно учитывать аспекты производительности и масштабируемости при проектировании архитектуры, особенно для крупных и сложных систем.

**3. Разработка специального программного обеспечения**

Специализированное программное обеспечение требует особых подходов:

* **Разработка встраиваемых систем:** Требует знаний в области аппаратного обеспечения и способности оптимизировать программное обеспечение для работы в ограниченных ресурсах.
* **Безопасность программного обеспечения:** Особенно критична в системах, связанных с финансами, здравоохранением и личными данными.
* **Соответствие стандартам и регулированиям:** Некоторые отрасли требуют соответствия специфическим стандартам и нормативам.

**4. Современные подходы и методологии**

* **Agile, Scrum, Kanban:** Гибкие методологии разработки, фокусирующиеся на итеративном процессе, адаптивности и непрерывном улучшении.
* **DevOps и непрерывная интеграция/доставка (CI/CD):** Процессы, позволяющие автоматизировать развертывание и обеспечивать высокую скорость и качество разработки.
* **Облачные технологии:** Использование облачных платформ (AWS, Azure, GCP) для разработки, тестирования и развертывания приложений.

**Заключение**

В области программной инженерии технологии и архитектура постоянно эволюционируют. Важно быть в курсе последних тенденций и лучших практик, чтобы создавать эффективные и надежные программные системы. Особое внимание стоит уделять специализированным областям разработки, где требования и ожидания могут значительно отличаться от общепринятых стандартов.

# 18 Концепции автоматизации процессов производства программного обеспечения на этапах его разработки и внедрения

# 19 Классификация информационных систем в программной инженерии

Классификация информационных систем в программной инженерии - это важный аспект для понимания и организации различных типов систем, с которыми могут сталкиваться инженеры. Вот основные категории:

1. **Транзакционные системы (Transaction Processing Systems, TPS)**: Эти системы управляют и обрабатывают большие объемы данных, связанные с транзакциями. Примеры включают системы управления заказами, системы биллинга, и банковские системы. Они обеспечивают высокую скорость обработки и надежность.
2. **Системы управления базами данных (Database Management Systems, DBMS)**: Это системы, которые предоставляют удобные и эффективные способы хранения, извлечения, обновления и управления данными. Примеры включают MySQL, PostgreSQL, и Oracle.
3. **Информационно-аналитические системы (Decision Support Systems, DSS)**: Эти системы помогают в принятии управленческих решений, предоставляя необходимые инструменты для анализа данных. Они могут включать аналитические модели, системы бизнес-аналитики и инструменты для обработки больших данных.
4. **Системы управления содержанием (Content Management Systems, CMS)**: CMS позволяют создавать, управлять и оптимизировать цифровой контент. Примеры включают WordPress, Drupal, и Joomla.
5. **Системы управления предприятием (Enterprise Resource Planning, ERP)**: ERP интегрируют различные подсистемы предприятия (финансы, управление запасами, человеческие ресурсы и т.д.) в единую информационную систему. Примеры включают SAP, Oracle ERP и Microsoft Dynamics.
6. **Клиентско-серверные системы**: Эти системы распределяют задачи между поставщиками услуг (серверами) и запросчиками услуг (клиентами). Примеры включают веб-серверы, базы данных и электронную почту.
7. **Облачные и распределенные системы**: Они используют ресурсы, распределенные по нескольким локациям (часто в облаке), для достижения эффективности и масштабируемости.
8. **Системы реального времени**: Системы, которые должны реагировать на события в определенные временные рамки. Примеры включают системы управления полетами, медицинские мониторинговые системы и автоматизированные системы управления производством.
9. **Интегрированные системы и IoT (Internet of Things)**: Эти системы сочетают в себе аппаратное и программное обеспечение для выполнения определенных функций, часто связанных с Интернетом вещей.
10. **Персональные и офисные приложения**: Включают стандартные приложения, такие как текстовые процессоры, электронные таблицы, системы электронной почты и т.д.

Эта классификация помогает понять, как различные типы информационных систем служат разным потребностям и как их можно использовать для достижения бизнес-целей и управления информационными процессами.

# 20 Управление конфигурациями программного обеспечения в программной инженерии

# 21 Автоматизация компиляции и компоновки программного обеспечения в программной инженерии

Автоматизация компиляции и компоновки (сборки) программного обеспечения играет ключевую роль в программной инженерии, поскольку она увеличивает эффективность процесса разработки, уменьшает вероятность ошибок, повышает повторяемость и облегчает интеграцию и доставку кода. Давайте подробно рассмотрим эти аспекты:

**1. Основные понятия**

**Компиляция**: Преобразование исходного кода, написанного на языке программирования высокого уровня, в машинный код или байт-код, который может быть выполнен на целевой машине или виртуальной машине.

**Компоновка (сборка)**: Процесс объединения различных частей программы (например, объектных файлов) в один исполняемый файл или библиотеку. Включает в себя решение зависимостей и адресации между модулями.

**2. Инструменты автоматизации**

**Системы сборки**: Инструменты вроде **make**, **Maven**, **Gradle**, **Ant**, **MSBuild**, которые автоматизируют процесс сборки, управляют зависимостями и обеспечивают консистентность сборок.

**Скрипты и утилиты**: Сценарии командной строки или специальные утилиты, которые автоматизируют рутинные задачи, например, очистка директорий от предыдущих сборок, копирование ресурсов, запуск тестов.

**Интеграция с контролем версий**: Автоматическая сборка при изменении в системе контроля версий, обычно с использованием CI/CD инструментов вроде Jenkins, GitLab CI, CircleCI.

**3. Преимущества автоматизации**

**Повышение эффективности**: Уменьшение времени, необходимого для сборки проекта, особенно в больших и сложных системах.

**Уменьшение ошибок**: Устранение человеческого фактора в процессе сборки, снижение вероятности возникновения ошибок из-за неправильной конфигурации или забытых шагов.

**Повторяемость и консистентность**: Гарантия того, что процесс сборки будет одинаковым на разных машинах и окружениях.

**Ускорение процесса разработки и тестирования**: Автоматическая сборка и развертывание позволяют разработчикам и тестировщикам быстрее получать последние версии ПО.

**Улучшение интеграции и доставки**: Упрощение процесса Continuous Integration (CI) и Continuous Delivery (CD), позволяя разработчикам интегрировать изменения чаще и с меньшими затратами времени.

**4. Лучшие практики**

**Конфигурация как код (Infrastructure as Code)**: Описывайте процессы сборки и развертывания с помощью кода для улучшения управляемости и версионирования.

**Модульность и управление зависимостями**: Правильное управление зависимостями и модульная архитектура помогают в облегчении сборки и обновлении компонентов.

**Регулярное обновление и поддержание системы сборки**: Следите за тем, чтобы инструменты и скрипты сборки были обновлены и соответствовали текущим требованиям проекта.

**Документация и стандартизация**: Поддержание актуальной документации по процессам сборки и стандартизация этих процессов облегчают новым членам команды вхождение в проект.

В заключение, автоматизация компиляции и компоновки является неотъемлемой частью современной программной инженерии, обеспечивающей более быструю, надежную и эффективную разработку программного обеспечения.

# 22 Технологии непрерывной интеграции комплексов программного обеспечения в программной инженерии

# 23 Непрерывная поставка и развертывание программного обеспечения в программной инженерии

**Непрерывная поставка (Continuous Delivery)**

**Определение**: Непрерывная поставка (CD) — это практика программной инженерии, при которой программное обеспечение автоматически собирается, тестируется и готовится к релизу в продакшн. Она стремится сделать выпуск новых функций и исправлений для пользователей как можно более быстрым и безопасным.

**Основные аспекты**:

1. **Автоматизация процесса сборки и тестирования**: Все изменения кода автоматически собираются, тестируются и готовятся к деплою. Используются инструменты CI (Continuous Integration) вроде Jenkins, GitLab CI, CircleCI и др.
2. **Управление конфигурациями и инфраструктурой**: Использование инструментов как Ansible, Terraform для обеспечения воспроизводимости и стандартизации сред.
3. **Тестирование в реальных условиях**: Включает в себя интеграционное, нагрузочное, и пользовательское тестирование.
4. **Функция "Feature Toggles"**: Возможность включать или отключать функции без необходимости деплоя новой версии.
5. **Возврат изменений (Rollbacks)**: Возможность быстро откатить изменения, если они вызывают проблемы.

**Непрерывное развертывание (Continuous Deployment)**

**Определение**: Непрерывное развертывание — это шаг дальше по сравнению с непрерывной поставкой. Здесь каждое изменение, прошедшее все этапы тестирования, автоматически развертывается в продакшн.

**Основные аспекты**:

1. **Полная автоматизация деплоя**: Не требуется ручного вмешательства для развертывания изменений в продакшн.
2. **Частые и мелкие релизы**: Позволяет минимизировать риски и упростить процесс обнаружения и исправления ошибок.
3. **Тесная интеграция с обратной связью пользователей**: Быстрое внедрение обратной связи в разработку.
4. **Мониторинг и логирование**: Необходимость в мощных инструментах мониторинга и логирования для отслеживания проблем в реальном времени.
5. **Культура DevOps**: Тесное взаимодействие между разработчиками и операционными командами для поддержки непрерывного цикла разработки.

**Важность в программной инженерии**

1. **Ускорение процесса разработки**: Непрерывные практики ускоряют получение обратной связи, позволяя быстрее улучшать и развивать продукт.
2. **Улучшение качества продукта**: Регулярное тестирование и интеграция помогают обнаруживать и исправлять проблемы на ранних этапах.
3. **Более высокая удовлетворенность клиентов**: Клиенты получают новые функции и исправления быстрее и чаще.
4. **Риск и управление изменениями**: Минимизация рисков благодаря мелким, контролируемым изменениям.
5. **Культурное влияние**: Способствует созданию культуры сотрудничества и постоянного улучшения.

Внедрение этих практик требует значительных усилий по настройке процессов и инструментов, а также культурных изменений внутри организации, но преимущества для бизнеса и качества продуктов делают эти усилия оправданными.

# 24 Управление качеством в программной инженерии. Показатели качества программного обеспечения

# 25 Виды испытаний программного обеспечения в соответствии с международными и отечественными стандартами и рекомендациями в программной инженерии

Тестирование программного обеспечения - это процесс, направленный на оценку качества и работоспособности программного продукта. Оно выполняется для выявления ошибок, проверки соответствия требованиям и обеспечения уверенности в надежности продукта. Тестирование может проводиться в соответствии с международными стандартами (например, ISO/IEC) и рекомендациями, а также с учетом отечественных стандартов в разных странах.

Вот основные виды тестирования программного обеспечения:

**1. Функциональное Тестирование**

Это тестирование на соответствие функциональным требованиям. Оно включает в себя проверку пользовательских сценариев, пользовательского интерфейса, баз данных, безопасности, клиент-серверных приложений и функциональности приложения.

**2. Нефункциональное Тестирование**

Оно оценивает аспекты программы, которые не связаны напрямую с определенной функцией, например:

* **Тестирование производительности:** Проверяет скорость, масштабируемость и другие показатели.
* **Тестирование нагрузки:** Проверяет поведение системы при ожидаемых и пиковых нагрузках.
* **Тестирование устойчивости (Stress Testing):** Проверяет пределы системы.
* **Тестирование безопасности:** Оценивает уязвимости и риски безопасности.
* **Тестирование совместимости:** Проверяет работу продукта в разных средах.

**3. Регрессионное Тестирование**

Оценивает, что новые изменения не влияют отрицательно на существующие функции продукта.

**4. Интеграционное Тестирование**

Проверяет взаимодействие между различными модулями или сервисами в приложении.

**5. Юнит-тестирование (Модульное Тестирование)**

Тестирует отдельные части программы, как правило, на уровне функций или методов.

**6. Акцептационное Тестирование**

Это финальная фаза тестирования, чтобы убедиться, что продукт готов к использованию. Это может включать в себя:

* **UAT (User Acceptance Testing):** Проверка конечными пользователями.
* **Beta Testing:** Распространение продукта ограниченному числу пользователей для внешней оценки.

**7. Тестирование Пользовательского Интерфейса**

Проверяет элементы интерфейса и взаимодействие с пользователем.

**8. Тестирование Доступности**

Проверяет, насколько легко продуктом могут пользоваться люди с различными ограничениями.

**Стандарты и Рекомендации**

* **ISO/IEC 25010:** Описывает модель качества программного продукта.
* **ISO/IEC 29119:** Серия стандартов для тестирования ПО.
* **IEEE 829:** Стандарт для документирования тестирования ПО.

Каждый из этих типов тестирования играет важную роль в процессе разработки программного обеспечения, помогая обеспечить его качество, надежность и удовлетворение требованиям пользователей и бизнеса.

# 26 Тестирование программного обеспечения в программной инженерии. Цели и виды тестирования. Основные фазы тестирования

# 27 Оптимизация и анализ исходных кодов программного обеспечения в программной инженерии

Оптимизация и анализ исходных кодов программного обеспечения являются ключевыми аспектами программной инженерии. Они направлены на повышение эффективности, надежности и поддерживаемости программных продуктов. Рассмотрим эти процессы более подробно.

**Оптимизация исходного кода**

Оптимизация кода заключается в улучшении его производительности и эффективности использования ресурсов (памяти, процессорного времени и т.д.) без изменения его функциональности. Оптимизация может включать следующие аспекты:

1. **Уменьшение времени выполнения**: Это достигается за счет улучшения алгоритмов, использования более быстрых операций и минимизации вызовов функций.
2. **Оптимизация использования памяти**: Это включает в себя устранение утечек памяти, оптимизацию структур данных и эффективное управление памятью.
3. **Улучшение читаемости и поддерживаемости кода**: Хотя это не влияет напрямую на производительность исполняемого кода, улучшение читаемости и поддерживаемости кода способствует более эффективной оптимизации и обслуживанию в будущем.
4. **Компиляторные оптимизации**: Использование продвинутых возможностей компиляторов для автоматической оптимизации кода.
5. **Профилирование производительности**: Использование инструментов профилирования для выявления "узких мест" в производительности и последующей их оптимизации.

**Анализ исходного кода**

Анализ исходного кода - это процесс изучения и оценки кода для выявления ошибок, уязвимостей, нарушений стилей кодирования и других проблем. Он может включать:

1. **Статический анализ**: Анализ кода без его выполнения для выявления потенциальных ошибок, уязвимостей безопасности и нарушений кодовых стандартов.
2. **Динамический анализ**: Тестирование программы во время ее выполнения для обнаружения ошибок, которые могут не быть очевидны при статическом анализе.
3. **Рефакторинг**: Процесс изменения внутренней структуры программы для улучшения ее читаемости и поддерживаемости, не изменяя ее внешнего поведения.
4. **Код-ревью**: Систематическое рассмотрение исходного кода коллегами для обнаружения ошибок и обмена знаниями.
5. **Использование линтеров и форматтеров**: Инструменты, которые автоматически проверяют код на соответствие стандартам кодирования и форматирования.

Обе эти области взаимосвязаны и важны для создания высококачественных, эффективных и надежных программных продуктов. Они требуют глубоких знаний в области алгоритмов, структур данных, принципов программирования и конкретных языков программирования.

# 28 Автоматизация тестирования программного обеспечения в программной инженерии

# 29 Существующие модели качества программного обеспечения в программной инженерии, их достоинства и недостатки

Существует множество моделей качества программного обеспечения, используемых в программной инженерии, каждая из которых имеет свои достоинства и недостатки. Рассмотрим некоторые из наиболее известных моделей:

1. **Модель МакКолла (McCall's Model)**
   * **Достоинства**: Основывается на трёх основных аспектах: эксплуатационных характеристиках, переходных характеристиках и характеристиках ревизии. Это обеспечивает комплексный подход к оценке качества ПО.
   * **Недостатки**: Может быть несколько устаревшей и не всегда полностью соответствовать современным требованиям к ПО, таким как гибкость и адаптивность.
2. **Модель Бума (Boehm's Model)**
   * **Достоинства**: Сильно фокусируется на пользовательском интерфейсе и удобстве использования, что особенно важно для потребительских приложений.
   * **Недостатки**: Может быть не так эффективна для оценки системного или встроенного программного обеспечения, где пользовательский интерфейс играет второстепенную роль.
3. **Модель ISO/IEC 9126**
   * **Достоинства**: Предлагает подробное и всестороннее описание качества ПО через шесть основных характеристик: функциональность, надежность, удобство использования, эффективность, обслуживаемость, переносимость.
   * **Недостатки**: Из-за своей сложности и детальности может быть трудоёмкой в реализации и требовать значительных ресурсов для полного анализа.
4. **Модель ISO/IEC 25010 (преемник ISO/IEC 9126)**
   * **Достоинства**: Обновленная и расширенная версия предыдущей модели, учитывает современные требования к ПО, включая удобство использования, безопасность, совместимость и другие.
   * **Недостатки**: Как и её предшественник, может быть сложной в применении, особенно в малых проектах.
5. **Модель CMMI (Capability Maturity Model Integration)**
   * **Достоинства**: Ориентирована не только на качество продукта, но и на процесс его разработки. Помогает организациям улучшать свои процессы разработки ПО.
   * **Недостатки**: Сложна в внедрении и поддержке, требует значительных усилий для достижения высоких уровней зрелости.
6. **Модель FURPS (Functionality, Usability, Reliability, Performance, Supportability)**
   * **Достоинства**: Проста в понимании и охватывает широкий спектр аспектов качества ПО.
   * **Недостатки**: Может быть не достаточно подробной для глубокой оценки определенных аспектов качества, таких как безопасность или совместимость.

Каждая из этих моделей имеет свои сильные и слабые стороны и может быть более или менее подходящей в зависимости от конкретных требований проекта и организации. Важно выбирать модель, которая лучше всего соответствует целям и контексту проекта.

# 30 Основные понятия, определения и стандарты программной инженерии

# 31 Жизненный цикл разработки программного обеспечения и его основные аспекты

Жизненный цикл разработки программного обеспечения (Software Development Life Cycle, SDLC) представляет собой процесс, который охватывает все этапы создания и сопровождения программного обеспечения. Основные аспекты и этапы жизненного цикла разработки программного обеспечения включают:

1. **Сбор и анализ требований:** На этом этапе команда разработчиков определяет цели и требования проекта, собирая информацию от заинтересованных сторон и пользователей. Это важно для понимания задач, которые должно выполнять ПО, и для определения ключевых функциональных и нефункциональных требований.
2. **Планирование:** На этапе планирования определяются ресурсы, необходимые для проекта (время, бюджет, персонал), а также риски и ограничения. Здесь также разрабатывается график работы и определяются ключевые вехи проекта.
3. **Дизайн и прототипирование:** На этом этапе команда разрабатывает архитектуру системы и создает детальные дизайн-спецификации. Включает в себя выбор стека технологий, проектирование баз данных, интерфейсов и прочих компонентов системы. Часто разработка начинается с создания прототипа для демонстрации ключевых функций и сбора обратной связи.
4. **Разработка и программирование:** Этот этап включает в себя реализацию программного обеспечения в соответствии с дизайн-спецификациями. Разработчики пишут код, интегрируют различные компоненты системы и реализуют необходимые функции.
5. **Тестирование:** На этом этапе ПО подвергается тщательному тестированию для выявления и исправления ошибок. Тестирование может включать в себя модульное тестирование, интеграционное, системное, приемочное и другие виды тестов.
6. **Развертывание:** После успешного тестирования программное обеспечение разворачивается в рабочей среде. Это может включать поэтапное развертывание, чтобы минимизировать риски для бизнеса.
7. **Сопровождение и поддержка:** После развертывания программное обеспечение требует постоянного обслуживания для исправления ошибок, обновления функций и улучшения производительности. Также важно обеспечивать поддержку пользователей.
8. **Обратная связь и итеративное улучшение:** Важно собирать обратную связь от пользователей и заинтересованных сторон для дальнейшего улучшения продукта. Жизненный цикл может быть итеративным, что означает повторение некоторых его фаз для улучшения и развития ПО.

SDLC может быть организован по-разному в зависимости от выбранной методологии разработки, такой как водопадная модель, Agile, SCRUM, Lean и другие. Каждая из этих методологий имеет свои особенности в плане управления процессами, вовлечения заинтересованных сторон и гибкости в управлении изменениями.

# 32 Проблема выбора модели жизненного цикла программного обеспечения и возможные пути ее решения

# 33 Тестирование программного обеспечения в программной инженерии. Техники тестирования и верификация программ

Конечно, давайте рассмотрим эти темы более подробно:

**Тестирование программного обеспечения в программной инженерии**

Тестирование программного обеспечения - это процесс анализа программного продукта для выявления различий между существующими и требуемыми условиями (то есть ошибок), а также для оценки его качества. Этот процесс включает в себя выполнение программного обеспечения с целью обнаружения ошибок и улучшения качества продукта.

Важность Тестирования:

1. **Обеспечение Качества:** Помогает гарантировать, что продукт работает так, как задумано, и удовлетворяет требованиям пользователя.
2. **Предотвращение Ошибок:** Выявляет и устраняет ошибки до того, как продукт попадет к конечному пользователю.
3. **Уменьшение Рисков:** Помогает предотвратить потенциальные риски, связанные с безопасностью, производительностью и надежностью.
4. **Улучшение Пользовательского Опыта:** Гарантирует, что пользовательский интерфейс интуитивно понятен и легок в использовании.

Типы Тестирования:

1. **Функциональное Тестирование:** Проверка специфических требований и функций.
2. **Нефункциональное Тестирование:** Оценка атрибутов системы, таких как производительность, удобство использования, надежность.
3. **Автоматизированное Тестирование:** Использование программного обеспечения для автоматизации выполнения тестов и сравнения фактических результатов с ожидаемыми.
4. **Ручное Тестирование:** Тестирование, выполняемое человеком без использования инструментов автоматизации.
5. **Интеграционное Тестирование:** Проверка взаимодействия между различными модулями или сервисами.
6. **Системное Тестирование:** Полное тестирование интегрированной системы для оценки её соответствия требованиям.
7. **Тестирование Приемки:** Оценка системы конечными пользователями для убеждения в том, что система готова к эксплуатации.

**Техники Тестирования и Верификация Программ**

Техники Тестирования:

1. **Черный Ящик:** Тестирование без учета внутренней структуры системы.
2. **Белый Ящик:** Тестирование с учетом внутренней структуры и работы программы.
3. **Серый Ящик:** Комбинация черного и белого ящика.
4. **Регрессионное Тестирование:** Проверка того, что недавние изменения в коде не повредили существующую функциональность.

Верификация Программ:

* **Статическая Верификация:** Включает в себя ревью кода, анализ кода на предмет ошибок без фактического выполнения программы.
* **Динамическая Верификация:** Включает в себя тестирование программы путем выполнения её на компьютере.

Верификация и тестирование являются ключевыми аспектами в обеспечении качества программного обеспечения и необходимы для выявления и исправления ошибок, а также для гарантии соответствия продукта установленным требованиям и стандартам.

# 34 Интеллектуальные информационные технологии в программной инженерии

# 35 Отечественные и зарубежные стандарты документирования программного обеспечения. Типы программной документации

Ваши вопросы касаются важных аспектов программной инженерии, и я постараюсь ответить на них максимально подробно.

**Отечественные и зарубежные стандарты документирования программного обеспечения**

Отечественные стандарты

В России и странах СНГ для документирования ПО часто используются ГОСТы (государственные стандарты), которые определяют требования к документации на разных этапах жизненного цикла программного продукта. Например, ГОСТ 19.101-77 описывает общие положения по программам для ЭВМ, а ГОСТ 19.102-77 устанавливает требования к программной документации.

Зарубежные стандарты

В международной практике широко используются стандарты ISO/IEC, например, серия ISO/IEC 12207, которая описывает процессы жизненного цикла программного обеспечения, включая аспекты документирования. Еще один важный стандарт - IEEE 1016, описывающий структуру документации по архитектуре программного обеспечения.

**Типы программной документации**

Программная документация делится на несколько основных типов, каждый из которых выполняет свои функции:

1. **Техническое задание (ТЗ)**: Этот документ определяет требования к программному продукту, его функциональность, ограничения и стандарты разработки.
2. **Проектная документация**: Включает в себя архитектуру системы, дизайн, схемы баз данных, диаграммы классов и последовательностей. Примеры включают функциональные спецификации, спецификации требований к программному обеспечению и т.д.
3. **Техническая документация**: Описывает процесс установки и настройки программного обеспечения, его системные требования и инструкции по использованию. Это включает руководства пользователя, справочные руководства и документацию API (прикладного программного интерфейса).
4. **Эксплуатационная документация**: Предназначена для IT-специалистов и включает в себя инструкции по обслуживанию, устранению неполадок, мониторингу и администрированию программного обеспечения.
5. **Документация по тестированию**: Включает в себя планы тестирования, тест-кейсы, сценарии и отчеты о тестировании. Эта документация используется для обеспечения качества и отслеживания исправления ошибок.
6. **Документация исходного кода**: Комментарии в коде, документирование API и разработка технической документации, которая помогает программистам понимать и поддерживать код.
7. **Отчеты о ходе разработки**: Включают отчеты о прогрессе, оценки, анализ рисков и другие документы, которые отслеживают ход выполнения проекта.

Каждый из этих типов документации играет важную роль в жизненном цикле разработки программного обеспечения, помогая обеспечить его качество, удобство поддержки и соответствие ожиданиям пользователя и требованиям заказчика.

# 36 Типы и назначение документации программного обеспечения в соответствии с Единой системой программной документации.

# 37 Комплекс стандартов на автоматизированные системы в программной инженерии

Комплекс стандартов на автоматизированные системы в программной инженерии охватывает широкий спектр аспектов, от разработки до поддержки программного обеспечения. Эти стандарты устанавливают общие принципы и практики, которым следуют инженеры и команды разработки для обеспечения качества, безопасности, эффективности и совместимости программных продуктов и систем. Основные стандарты включают:

1. **ISO/IEC 12207 (Информационная технология – Процессы жизненного цикла программного обеспечения)**: Этот стандарт описывает процессы, связанные с жизненным циклом программного обеспечения, включая планирование, разработку, эксплуатацию и поддержку. Он предоставляет общую терминологию и процессы, которые помогают организациям управлять своими проектами по разработке программного обеспечения.
2. **ISO/IEC 25010 (Системы и программное обеспечение – Модели качества)**: Этот стандарт определяет модель качества для оценки качества программного обеспечения и систем. Он включает различные характеристики, такие как функциональность, надежность, удобство использования, эффективность, сопровождаемость и переносимость.
3. **IEEE 830-1998 (Стандарт IEEE для спецификаций требований к программному обеспечению)**: Этот стандарт описывает рекомендации по составлению хороших спецификаций требований к программному обеспечению. Он включает в себя методы идентификации и документирования требований, что является ключевым аспектом успеха проекта по разработке программного обеспечения.
4. **CMMI (Capability Maturity Model Integration)**: Хотя это не формальный стандарт ISO или IEEE, CMMI широко используется в индустрии для оценки зрелости процессов организации в области разработки программного обеспечения. Он включает ряд уровней зрелости, от начального до оптимизирующего, каждый из которых описывает различные процессы и практики.
5. **Agile/Scrum методологии**: Хотя Agile и Scrum не являются формальными стандартами, они представляют собой широко принятые практики в разработке программного обеспечения. Они ориентированы на итеративную разработку, гибкость, вовлечение клиента и адаптивное планирование.
6. **ISO/IEC 27001 (Информационная безопасность)**: Этот стандарт не специфичен только для программной инженерии, но он крайне важен в контексте разработки безопасного программного обеспечения. Он устанавливает требования к системе управления информационной безопасностью.

Эти стандарты и практики взаимодействуют друг с другом, обеспечивая комплексный подход к управлению проектами по разработке программного обеспечения, поддержке качества и эффективности, а также к обеспечению безопасности и соблюдению нормативных требований.

# 38 Технологии формирования и ведения баз знаний при разработке программного обеспечения в программной инженерии

# 39 Классификация операционных систем в программной инженерии. Операционные системы на базе GNU/Linux. Операционные системы специального назначения в промышленном производстве программного обеспечения

**1. Классификация операционных систем в программной инженерии**

Операционные системы (ОС) можно классифицировать по различным критериям:

1. **Тип пользователя и способ использования:**
   * **Однопользовательские и многопользовательские**: Однопользовательские ОС предназначены для одного пользователя (например, Windows 10 Home). Многопользовательские ОС позволяют одновременно работать многим пользователям (например, UNIX, Linux).
2. **Количество выполняемых задач:**
   * **Однозадачные и многозадачные**: Однозадачные ОС могут выполнять только одну задачу за раз, в то время как многозадачные ОС позволяют выполнять множество задач одновременно (например, Windows, Linux).
3. **Тип управления ресурсами:**
   * **Реального времени**: Предназначены для приложений, где важна быстрая и предсказуемая реакция на внешние события (например, RTOS).
   * **Общего назначения**: Предназначены для широкого спектра задач и приложений.
4. **Тип обработки:**
   * **Централизованные и распределенные**: Централизованные ОС управляют всеми ресурсами из одного места, в то время как распределенные ОС используют ресурсы, распределенные по сети.
5. **По архитектуре:**
   * **Монолитные, микроядерные, гибридные**: Монолитные ОС имеют одно большое ядро, микроядерные разделяют функции на более мелкие, независимые модули, а гибридные сочетают оба подхода.

**2. Операционные системы на базе GNU/Linux**

GNU/Linux – это термин, используемый для описания семейства свободных и открытых операционных систем, которые основаны на ядре Linux и инструментах GNU. Основные особенности:

* **Открытый исходный код**: Позволяет пользователям изучать, изменять и распространять код.
* **Многозадачность и многопользовательская среда**: Поддерживаются одновременно работа многих пользователей и выполнение множества задач.
* **Безопасность и стабильность**: Является одной из самых безопасных ОС благодаря своей архитектуре и активному сообществу.
* **Настройка и гибкость**: Может быть адаптирована под различные потребности пользователей и организаций.

**3. Операционные системы специального назначения в промышленном производстве программного обеспечения**

Операционные системы специального назначения разработаны для удовлетворения конкретных требований промышленного производства ПО:

* **Встраиваемые системы (Embedded Systems)**: Предназначены для управления специализированным оборудованием (например, системы управления на производственных линиях).
* **RTOS (Real-Time Operating Systems)**: Используются в системах, где критически важны время отклика и предсказуемость (например, в авиационных и автомобильных системах).
* **Системы управления базами данных**: Оптимизированы для работы с большими объемами данных и обеспечения быстрого доступа к ним.
* **Сетевые ОС**: Специализируются на управлении и координации работы в больших сетях.

Эти ОС обычно характеризуются высокой надежностью, стабильностью, способностью работать в реальном времени и специализированной функциональностью для конкретных задач.

# 40 Технологии виртуализации и контейнеризации в операционных системах на этапах разработки и эксплуатации программного обеспечения