**ОТЧЕТ**

**о выполнении лабораторной работы №6**

**Дисциплина «МДК 03.01 Моделирование и анализ программного обеспечения»**

**Тема: «План-конспект»**

**Специальность: 09.02.07 Информационные системы и программирование**

**Квалификация: Специалист по информационным системам**

Выполнил: Стадников М.А.

Студент группы: ИС50-2-22

Проверил:

Преподаватель: Кретова Е.М.

Дата: «\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Порядок выполнения работы**

1. Жизненный цикл программного обеспечения

1.1. Определение «Жизненный цикл ПО»

Жизненный цикл программного обеспечения (ЖЦПО) — это последовательность этапов, через которые проходит программный продукт от его первоначальной идеи до завершения эксплуатации и утилизации.

1.2. Этапы разработки жизненного цикла

Анализ требований: Сбор и анализ требований пользователей и заинтересованных сторон.

Проектирование: Создание архитектуры и дизайна системы.

Разработка: Написание кода и создание программного продукта.

Тестирование: Проверка и валидация программного обеспечения на наличие ошибок и соответствие требованиям.

Внедрение: Установка и настройка программного обеспечения в рабочей среде.

Эксплуатация и поддержка: Поддержка пользователей и исправление ошибок.

Выведение из эксплуатации: Завершение работы с программным продуктом и его утилизация.

1.3. Каскадная модель жизненного цикла программного обеспечения

Описание: Модель, в которой каждый этап разработки следует за предыдущим без возврата. Подразумевает четкую последовательность и завершение одного этапа перед началом следующего.

**Плюсы**:

**Простота и понятность**: Легко понять и объяснить, что делает модель подходящей для небольших проектов.

**Четкие этапы**: Каждый этап имеет четкие цели и результаты, что упрощает управление проектом.

**Легкость в планировании**: Этапы можно легко планировать и оценивать по времени.

Минусы:

**Невозможность вернуться**: Изменения на поздних этапах могут быть сложными и дорогостоящими.

**Неэффективность для больших проектов**: Модели требуют жесткой структуры, что может быть проблемой для сложных и крупных проектов.

**Ограниченная гибкость**: Неспособность адаптироваться к изменениям требований в процессе разработки.

1.4. Инкрементная модель жизненного цикла программного обеспечения

**Описание**: Модель, в которой система разрабатывается и поставляется по частям (инкрементам). Каждая новая версия добавляет функциональность к предыдущей.

**Плюсы**:

**Быстрая поставка**: Возможность быстрого выпуска первых версий продукта для получения обратной связи.

**Гибкость**: Легко вносить изменения и добавлять новые функции в процессе разработки.

**Уменьшение рисков**: Позволяет тестировать и проверять функциональность на каждом этапе.

**Минусы**:

**Сложность управления**: Требует хорошего планирования и управления для обеспечения согласованности между инкрементами.

**Потенциальные проблемы с интеграцией**: Каждая новая версия может привести к интеграционным проблемам с предыдущими.

**Необходимость постоянного тестирования**: Требует постоянного тестирования и контроля качества на каждом этапе.

1.5. Спиральная модель жизненного цикла программного обеспечения

**Описание**: Модель, которая сочетает элементы каскадной и инкрементной моделей. Разработка проходит через циклы (спирали), где каждый цикл включает планирование, анализ рисков, разработку и тестирование.

**Плюсы**:

**Управление рисками**: Обеспечивает регулярный анализ рисков на каждом цикле.

**Гибкость и адаптивность**: Позволяет вносить изменения и корректировки на каждом этапе.

**Итеративный подход**: Постепенное улучшение продукта на основе обратной связи.

Минусы:

**Сложность**: Требует высокой квалификации команды для управления и выполнения каждого цикла.

**Высокие затраты:** Может быть более затратной из-за постоянного анализа и тестирования.

**Неопределенность**: Результаты могут быть непредсказуемыми из-за постоянных изменений.

1.6. V-образная модель жизненного цикла программного обеспечения

Описание: Модель, в которой акцент делается на валидацию и верификацию. Каждый этап разработки имеет соответствующий этап тестирования.

**Плюсы**:

**Четкая связь между этапами**: Каждый этап разработки связан с соответствующим этапом тестирования, что улучшает качество.

**Раннее выявление ошибок**: Тестирование на каждом этапе позволяет быстро находить и исправлять ошибки.

**Легкость в управлении:** Структурированный подход облегчает управление проектом.

**Минусы**:

**Жесткость**: Невозможность вносить изменения на поздних этапах разработки.

Высокие затраты на тестирование: Требует значительных ресурсов для тестирования на каждом этапе.

**Не подходит для гибких проектов**: Неэффективна для проектов, где требования могут изменяться в процессе разработки.

1.7. Итеративная модель жизненного цикла программного обеспечения

**Описание**: Модель, в которой разработка проходит через несколько итераций, позволяя команде улучшать продукт на основе обратной связи на каждом этапе.

**Плюсы**:

**Гибкость**: Позволяет легко вносить изменения и улучшения на основе полученной обратной связи.

**Быстрая поставка**: Возможность выпуска промежуточных версий для тестирования и получения отзывов.

**Улучшение качества**: Постоянное тестирование и улучшение продукта на каждой итерации.

**Минусы**:

**Сложность управления**: Требует хорошего управления для координации между итерациями.

**Риск недостаточной документации**: Быстрая итерация может привести к нехватке документации.

**Возможные проблемы с интеграцией**: Частые изменения могут вызвать сложности при интеграции новых функций.

2.1. Методология Waterfall

Описание: Методология Waterfall (водопадная модель) представляет собой классический подход к разработке программного обеспечения, где проект проходит через четко определенные последовательные этапы. Каждый этап должен быть завершен перед переходом к следующему, что создает линейный и структурированный процесс.

Этапы:

Анализ требований: Сбор и документирование всех требований к системе.

Проектирование: Создание архитектуры системы и проектирование интерфейсов.

Разработка: Программирование и реализация системы.

Тестирование: Проверка системы на наличие ошибок и соответствие требованиям.

Внедрение: Развертывание системы в рабочей среде.

Эксплуатация: Поддержка и обслуживание системы.

Вывод из эксплуатации: Завершение проекта и удаление системы.

Преимущества:

Простота и понятность структуры, что облегчает управление проектом.

Четкие сроки и результаты для каждого этапа.

Недостатки:

Невозможность вернуться к предыдущим этапам, что затрудняет внесение изменений.

Высокая стоимость изменений на поздних этапах.

Использование: Waterfall подходит для проектов с четкими требованиями и небольшим количеством изменений. Часто используется в государственных и крупных корпоративных проектах, где важна документация и соблюдение регламентов.

2.2. Методология Agile

Описание: Agile (гибкая методология) — это подход к разработке программного обеспечения, который акцентирует внимание на гибкости, адаптивности и быстром реагировании на изменения. Agile подразумевает регулярные итерации, которые позволяют команде быстро адаптироваться к новым требованиям и получать обратную связь от пользователей.

Основные принципы:

Ориентация на клиента: приоритет отдается удовлетворению потребностей клиентов.

Итеративная разработка: работа делится на короткие циклы (итерации), обычно от одной до четырех недель.

Постоянное взаимодействие: регулярные встречи и обсуждения между членами команды.

Преимущества:

Высокая гибкость и возможность быстрого реагирования на изменения.

Увеличение вовлеченности клиентов и пользователей в процесс разработки.

Недостатки:

Меньшая предсказуемость сроков и бюджета по сравнению с более традиционными подходами.

Необходимость постоянного взаимодействия и коммуникации.

Использование: Agile используется в динамичных проектах, где требования могут меняться, таких как стартапы и IT-компании, разрабатывающие новые продукты.

2.3. Методология Kanban

Описание: Kanban — это методология управления проектами, основанная на визуализации рабочего процесса и управлении задачами с помощью канбан-досок. Этот подход позволяет командам видеть текущие задачи и их статус, что способствует более эффективному управлению потоком работы.

Основные элементы:

Канбан-доска: Визуальный инструмент, на котором отображаются задачи в различных стадиях выполнения (например, "Запланировано", "В процессе", "Завершено").

Лимиты на работу в процессе (WIP): Ограничения на количество задач, которые могут находиться в работе одновременно, чтобы избежать перегрузки команды.

Преимущества:

Улучшение прозрачности и видимости рабочего процесса.

Упрощение управления задачами и приоритизация работы.

Недостатки:

Может быть сложным для внедрения в командах с устоявшимися процессами.

Не всегда подходит для проектов с четкими сроками и жесткими требованиями.

Использование: Kanban широко используется в производственных и IT-компаниях для оптимизации процессов и повышения эффективности работы команд.

2.4. Методология Scrum

Описание: Scrum — это итеративная методология разработки, которая делит проект на короткие временные отрезки, называемые спринтами (обычно от одной до четырех недель). Scrum включает регулярные встречи, такие как планирование спринта, ежедневные стендапы и ретроспективы, что позволяет команде оценивать прогресс и вносить необходимые изменения.

Основные роли:

Продуктовый владелец (Product Owner): Определяет требования и приоритеты задач.

Команда разработки: Работает над выполнением задач в рамках спринта.

Scrum-мастер: Обеспечивает соблюдение принципов Scrum и помогает команде устранять препятствия.

Преимущества:

Быстрая поставка функциональности и возможность получения обратной связи.

Улучшение командной работы и взаимодействия.

Недостатки:

Требует высокой степени самоорганизации и ответственности от команды.

Может быть сложно внедрить в организации с традиционными подходами к управлению проектами.

Использование: Scrum подходит для проектов с неопределенными требованиями и высокими темпами изменений, таких как разработка ПО, веб-приложений и стартапов.2.5. Локальные системы контроля версий

Системы, которые хранят версии кода на локальных машинах разработчиков (например, Git).

2.6. Централизованные системы контроля версий

Системы, в которых все версии кода хранятся на центральном сервере (например, Subversion).

2.7. Распределённые системы контроля версий

Системы, которые позволяют каждому разработчику иметь полную копию репозитория (например, Git), что облегчает работу в команде и управление версиями.

3. Механизмы и контроль внесения изменений в код

3.1. Прототип

Предварительная версия продукта, созданная для демонстрации концепции и получения отзывов.

3.2. Альфа версия

Первая версия программного продукта, предназначенная для внутреннего тестирования и выявления основных ошибок.

3.3. Бета версия

Версия, выпущенная для ограниченного круга пользователей с целью тестирования и получения обратной связи перед окончательной поставкой.

3.4. Окончательная поставка

Завершенная версия программного продукта, готовая к развертыванию и использованию конечными пользователями.

Эта структура поможет вам систематизировать информацию и подготовить материал по каждому из пунктов.