МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению лабораторных работ
по дисциплине «Современные методологии инженерии программного обеспечения» для студентов направления подготовки
09.04.04 Программная инженерия Направленность (профиль)
«Разработка, мониторинг и управление жизненным циклом инженерных систем»

Ставрополь 2026

Лабораторная работа 1.
«Составление User Stories и Acceptance Criteria»

1. Цель работы

Приобретение практических навыков:

- Формулирования требований к программному обеспечению в формате User Stories;
- Разработки критериев приемки (Acceptance Criteria);
- Приоритизации пользовательских историй;
- Декомпозиции функциональных требований в формат, пригодный для планирования спринта.

2. Теоретическое введение

2.1. Что такое User Story?

User Story (Пользовательская история) — это краткое описание функциональности, полезной для пользователя, сформулированное на языке бизнеса, понятном как заказчику, так и команде разработки.

Классический шаблон (Connextra Template):

«Как *[роль пользователя]***
Я хочу *[действие/функциональность]***
Чтобы *[достичь цели / получить ценность]»***

Пример:

Как **зарегистрированный пользователь**,
я хочу **восстановить пароль по электронной почте**,
чтобы **получить доступ к аккаунту в случае его утраты.**

2.2. Критерии качественной User Story (INVEST)

Хорошая пользовательская история должна соответствовать критериям **INVEST**:

Критерий	Описание
I — Independent	Независима от других историй (может быть реализована в любом порядке)
N — Negotiable	Обсуждаема (не жестко зафиксированный контракт, а предмет диалога)
V — Valuable	Ценна для пользователя или бизнеса
E — Estimable	Оцениваема (команда может оценить трудоемкость)

S — Small	Достаточно мала для реализации в рамках одного спринта
T — Testable	Проверяема (существуют четкие критерии приемки)

2.3. Acceptance Criteria (Критерии приемки)

Acceptance Criteria (Критерии приемки) — это набор четких, измеримых условий, выполнение которых подтверждает, что пользовательская история реализована корректно и готова к приемке.

Форматы описания:

1. **Формат «Сценарий» (Gherkin / Behavior-Driven Development):**

text

Scenario: Успешное восстановление пароля

Given пользователь находится на странице входа

When пользователь нажимает «Забыли пароль?» и вводит email

Then система отправляет письмо со ссылкой для сброса пароля

2. **Формат «Чек-лист»:**

- Ссылка для сброса пароля действительна в течение 24 часов
- После сброса пароля пользователь получает уведомление на email
- Новый пароль соответствует политике безопасности (мин. 8 символов, цифра, спецсимвол)

3. Задание

3.1. Исходные данные

Вы — Product Owner (или аналитик) в команде разработки. Компания заказала разработку мобильного приложения **«FitTrack»** — трекера здоровья и физической активности.

Описание продукта:

Приложение позволяет пользователям отслеживать физическую активность, вести дневник питания, ставить цели по тренировкам и делиться достижениями с друзьями.

Стейкхолдеры и их потребности:

1. **Пользователь (новичок):** хочет быстро разобраться в интерфейсе и начать отслеживать базовые метрики (шаги, калории).
2. **Пользователь (продвинутый):** хочет ставить сложные цели (например, пробежать марафон), анализировать прогресс в графиках, синхронизироваться с Apple Watch/Garmin.

3. **Тренер:** хочет просматривать прогресс своих клиентов и оставлять комментарии.

4. **Бизнес (владелец продукта):** хочет удерживать пользователей (социальные функции, ачивки) и монетизировать приложение (подписка на премиум-функции).

3.2. Этапы выполнения

Этап 1. Составление User Stories (20 баллов)

Составьте **5 пользовательских историй** по шаблону «Как… я хочу… чтобы…». Истории должны покрывать следующие функциональные области (по одной на каждую область):

№	Функциональная область	Целевая роль
1	Регистрация и онбординг	Новый пользователь
2	Отслеживание активности	Активный пользователь
3	Социальное взаимодействие	Пользователь
4	Управление целями	Продвинутый пользователь
5	Панель тренера	Тренер

Требования к каждой User Story:

- Использовать правильный шаблон формулировки.
- Указать роль и цель, понятную с точки зрения бизнес-ценности.
- Проверить соответствие критерию **INVEST** (самопроверка).

Этап 2. Разработка Acceptance Criteria (20 баллов)

Для **двух любых** из составленных User Stories разработайте подробные критерии приемки в двух форматах:

1. **Формат Gherkin (сценарий)** — минимум 2 сценария (Happy Path + хотя бы один альтернативный/исключительный путь).
2. **Формат чек-листа** — минимум 4 пункта, включая:
   - Функциональные требования;
   - Требования к безопасности (если применимо);
   - Требования к производительности (если применимо);
   - Условия граничных значений.

Этап 3. Приоритизация и оценка (10 баллов)

Для всех 5 составленных User Stories выполните:

1. **Приоритизацию методом MoSCoW:**

- **Must have** — критически важно для MVP (Minimum Viable Product)
- **Should have** — важно, но может быть реализовано позже
- **Could have** — приятно иметь, но не критично
- **Won't have** — не входит в текущий релиз

2. **Оценку трудоемкости в Story Points** (используя последовательность Фибоначчи: 1, 2, 3, 5, 8, 13):
   - Обоснуйте, почему каждая история получила именно такую оценку.

Этап 4. Формулировка Definition of Done (5 баллов)

Составьте **Definition of Done (DoD)** — чек-лист критериев, которым должна соответствовать каждая завершенная User Story в вашей команде. Минимум 6 пунктов, включая:

- Критерии по качеству кода;
- Критерии по тестированию;
- Критерии по документации;
- Критерии по развертыванию.

4. Формат отчета

Отчет оформляется в виде текстового документа (Markdown, Word или PDF) и должен содержать:

1. **Титульный лист** (название работы, ФИО, группа, вариант).
2. **Раздел 1. User Stories** — таблица с 5 историями.
3. **Раздел 2. Acceptance Criteria** — детальное описание для двух историй (сценарии Gherkin + чек-лист).
4. **Раздел 3. Приоритизация и оценка** — таблица с результатами MoSCoW и Story Points + обоснование.
5. **Раздел 4. Definition of Done** — чек-лист команды.
6. **Вывод** — краткий анализ полученных навыков, сложностей, с которыми столкнулись при формулировке требований.

5. Пример выполнения (для справки)

User Story (Пример для области «Регистрация»)

Как новый пользователь, **я хочу** зарегистрироваться через Google-аккаунт, **чтобы** не запоминать еще один пароль и ускорить вход в приложение.

INVEST-проверка:

- ✅Independent — не зависит от других историй
- ✅Valuable — реальная ценность для пользователя (удобство)
- ✅Testable — проверяется наличием корректной аутентификации

Acceptance Criteria для истории выше

Формат Gherkin:

text

```
Scenario: Успешная регистрация через Google
  Given пользователь открыл приложение FitTrack
  When пользователь выбирает «Войти через Google»
  And выбирает аккаунт из предложенного списка
  Then система создает новый профиль пользователя
  And перенаправляет на экран онбординга

Scenario: Ошибка при отмене авторизации
  Given пользователь открыл приложение FitTrack
  When пользователь выбирает «Войти через Google»
  And нажимает «Отмена» в окне выбора аккаунта
  Then система возвращает пользователя на экран входа
  And отображает сообщение «Авторизация отменена»
```

Формат чек-листа:

- При первом входе через Google создается новый профиль с извлечением имени и email из аккаунта Google
- При повторном входе через тот же Google-аккаунт восстанавливается существующий профиль
- Данные пользователя сохраняются в защищенном хранилище (токен OAuth 2.0)
- Время ответа при авторизации не превышает 3 секунд при стабильном интернет-соединении
- При отсутствии интернет-соединения отображается понятное сообщение об ошибке

7. Вопросы для защиты

1. В чем разница между User Story и традиционным техническим заданием (ТЗ)?
2. Почему критерии INVEST важны для гибкой разработки?
3. Какие проблемы могут возникнуть, если Acceptance Criteria слишком детализированы? А если слишком размыты?
4. Как связаны User Stories и бэклог продукта (Product Backlog)?
5. Кто должен участвовать в написании и уточнении User Stories?

8. Рекомендуемые инструменты

Для выполнения работы рекомендуется использовать:

- **Miro / Mural** — для визуализации карты пользовательских историй (User Story Mapping)
- **Jira / Trello** — для отработки навыков ведения бэклога
- **Markdown** — для оформления сценариев Gherkin в читаемом формате

Лабораторная работа 2
«Настройка Kanban-доски в Jira с ограничением WIP»

1. Цель работы

Приобретение практических навыков:

- Создания и настройки Kanban-доски в Jira;
- Установки ограничений на незавершенную работу (Work in Progress, WIP);
- Визуализации рабочего процесса команды разработки;
- Анализа эффективности потока задач с использованием WIP-лимитов.

2. Теоретическое введение

2.1. Что такое WIP-лимиты?

WIP (Work in Progress) Limit — это ограничение максимального количества задач, которые могут одновременно находиться на определенном этапе рабочего процесса .

В Kanban WIP-лимиты являются одним из ключевых механизмов, позволяющих:

- Предотвратить перегрузку команды;
- Сократить время выполнения задач (cycle time);
- Выявить узкие места (bottlenecks) в процессе разработки;
- Снизить потери от переключения контекста .

2.2. Почему WIP-лимиты важны?

Эффект	Описание
Снижение многозадачности	Команда сосредотачивается на завершении начатых задач, а не на старте новых
Выявление блокеров	Когда задачи накапливаются в одной колонке, становится очевидно, где возникает затор
Ускорение поставки	Уменьшение количества "почти готовых" задач ускоряет доставку ценности клиенту
Культура завершения	Формирует привычку доводить задачи до конца, а не распыляться

2.3. Как работают WIP-лимиты в Jira?

В Jira WIP-лимиты настраиваются для каждой колонки Kanban-доски. При превышении установленного лимита:

- Фон колонки окрашивается в **красный цвет**;
- Отображается счетчик превышения (например, "3/2" — 3 задачи при лимите 2) .

Jira **не отправляет автоматических уведомлений** при превышении лимита, но визуальное выделение служит сигналом для команды о необходимости разобрать накопившиеся задачи .

2.4. Рекомендации по установке WIP-лимитов

Для новых команд:

1. Сначала поработайте без ограничений 2-3 спринта;
2. Отследите среднее количество задач в каждой колонке;
3. Установите лимит на 10-20% выше среднего значения .

Базовые правила:

- Общий лимит на колонку **"In Progress"** часто устанавливают меньше количества разработчиков в команде (например, 3 лимит для команды из 4 человек), чтобы стимулировать помощь коллегам ;
- Для колонки **"Done"** лимит не устанавливается;
- Если лимит постоянно превышается — это сигнал о проблеме в процессе, а не повод увеличить лимит .

3. Задание

3.1. Исходные данные

Вы — Scrum-мастер/Agile-коуч в команде разработки мобильного приложения **«FitTrack»** (знакомого по предыдущей лабораторной работе). Команда состоит из **5 человек**: 3 разработчика, 1 тестировщик, 1 аналитик.

Текущие проблемы команды:

- Задачи накапливаются в колонке "Code Review" — разработчики не успевают проверять код друг друга;
- Часто случаются авралы в конце спринта, хотя загрузка в начале была нормальной;
- Разработчики берут по 3-4 задачи одновременно, переключаясь между ними;
- Тестировщик простаивает в середине спринта, а в конце перегружен.

3.2. Этапы выполнения

Этап 1. Создание Kanban-доски (15 баллов)

Выполните в Jira (облачная версия или локальная установка):

1. **Создайте новый проект** типа "Kanban" (или создайте Kanban-доску в существующем проекте).

2. **Настройте колонки** в соответствии со следующим рабочим процессом:

Колонка	Назначение
Backlog	Бэклог продукта (запланированные, но не взятые в работу задачи)
To Do	Задачи, готовые к выполнению (все требования согласованы)
In Progress	Задачи, находящиеся в активной разработке
Code Review	Задачи, ожидающие проверки кода
Testing	Задачи, переданные на тестирование
Done	Завершенные задачи

3. **Создайте тестовые задачи** (минимум 8-10) — используйте User Stories из предыдущей лабораторной работы (приложение «FitTrack»).

Этап 2. Настройка WIP-лимитов (20 баллов)

Установите WIP-лимиты для колонок согласно следующей таблице (обоснуйте выбор в отчете):

Колонка	WIP-лимит	Обоснование
Backlog	— (без лимита)	
To Do	8	
In Progress	3	
Code Review	2	
Testing	3	
Done	— (без лимита)	

Процедура настройки WIP-лимитов в Jira:

1. Перейдите в настройки доски: ... → **Board settings** → **Columns** .

2. Для каждой колонки нажмите на значок карандаша (редактирование).

3. В поле **"WIP Limit"** укажите максимальное количество задач.

4. Сохраните изменения кнопкой **Save Changes** .

Этап 3. Визуальное наблюдение за превышением лимитов (15 баллов)

Создайте сценарий, при котором происходит превышение WIP-лимитов:

1. Переместите достаточное количество задач в колонку **"In Progress"**, чтобы превысить установленный лимит (например, 4 задачи при лимите 3).

2. Переместите достаточное количество задач в колонку **"Code Review"**, чтобы превысить лимит (3 задачи при лимите 2).

Зафиксируйте в отчете:

- Скриншот доски с визуальным отображением превышения лимитов;
- Описание, как именно Jira сигнализирует о нарушении (цвет фона, счетчик).

Этап 4. Добавление Swimlanes (15 баллов)

Для улучшения видимости работы добавьте **горизонтальные разделители (Swimlanes)** по следующим критериям :

1. **Swimlane по приоритету:** High Priority / Medium Priority / Low Priority.

2. **ИЛИ Swimlane по назначению:** по каждому разработчику (Assignees).

Настройка Swimlanes:

1. Перейдите: **Board settings → Swimlanes**.

2. Выберите тип разделения (например, "Queries" для JQL-фильтра по приоритету).

3. Для разделения по приоритету используйте JQL: `priority = High` и т.д.

4. Сохраните изменения .

Этап 5. Автоматизация уведомлений (опционально, +5 баллов)

Настройте автоматическое уведомление (Slack или email) при превышении WIP-лимита в колонке "Code Review":

1. Перейдите: **Project settings → Automation**.

2. Создайте новое правило с триггером: When an issue transitions to "Code Review".

3. Добавьте условие: If the number of issues in "Code Review" > WIP limit.

4. Добавьте действие: Send Slack message или Send email .

Этап 6. Анализ эффективности (15 баллов)

Ответьте на следующие вопросы (письменно в отчете):

1. **Выявление узких мест:** Какая колонка, по вашему мнению, станет первым узким местом при установленных лимитах? Почему?
2. **Командная динамика:** Как WIP-лимиты повлияют на поведение разработчиков, которые привыкли брать много задач одновременно?
3. **Профилактика простоев:** Как лимиты помогут загрузить тестировщика равномерно в течение спринта?
4. **Метрики:** Какие метрики (встроенные в Jira) можно использовать для оценки эффективности WIP-лимитов? (Укажите: **Cumulative Flow Diagram**, **Control Chart**)

4. Формат отчета

Отчет оформляется в виде текстового документа и должен содержать:

1. **Титульный лист** (название работы, ФИО, группа).
2. **Раздел 1. Скриншот созданной доски** с перечислением всех колонок.
3. **Раздел 2. Таблица WIP-лимитов** с обоснованием каждого выбора.
4. **Раздел 3. Скриншот превышения лимитов** (красная подсветка колонок) + описание реакции Jira.
5. **Раздел 4. Скриншот доски с Swimlanes** (по приоритету или по назначению).
6. **Раздел 5. Код автоматизации** (если выполняли) или описание шагов настройки.
7. **Раздел 6. Ответы на вопросы анализа** (4 вопроса из Этапа 6).
8. **Вывод** — что дало использование WIP-лимитов, какие проблемы команды они помогают решить.

5. Пример ожидаемого результата

Скриншот колонки с превышением WIP-лимита:

text

```
+-------------------------------------------------+
| IN PROGRESS (WIP: 3)                 [4] ● |
| +---------------+                          |
| | FIT-001: Story A |                       |
```

	FIT-002: Story B	
	FIT-003: Story C	
	FIT-004: Story D	← 4-я задача (превышение)
+---------------+
+---+

При превышении лимита Jira подсвечивает колонку красным фоном и отображает счетчик "4/3" .

6. Критерии оценки

Критерий	Максимальный балл
Корректное создание Kanban-доски и колонок	15
Обоснованность установленных WIP-лимитов	20
Фиксация и описание превышения лимитов (скриншот)	15
Настройка Swimlanes	15
Качество анализа эффективности WIP-лимитов	15
Структура отчета, оформление, выводы	10
Итого	**90 (+10 опционально)**

7. Вопросы для защиты

1. Что такое WIP-лимит и почему он является ключевым элементом Kanban?
2. Какие визуальные индикаторы Jira показывают превышение WIP-лимита?
3. Почему не рекомендуется устанавливать WIP-лимиты сразу после создания доски?
4. Как WIP-лимиты помогают выявлять узкие места в процессе разработки?
5. В чем разница между **Lead Time** и **Cycle Time**? Какая метрика более важна для Kanban?
6. Почему для колонки "Done" обычно не устанавливают WIP-лимит?
7. Какова роль WIP-лимитов в формировании культуры "завершения" (culture of done)?

8. Дополнительные материалы

8.1. Полезные ссылки

- Официальная документация Atlassian по WIP-лимитам: https://www.atlassian.com/ru/agile/kanban/wip-limits
- Cumulative Flow Diagram (CFD) — ключевой отчет для анализа потока задач

8.2. Типичные анти-паттерны при работе с WIP-лимитами

Анти-паттерн	Почему это плохо
Увеличение лимита, чтобы команда "не билась в него"	Маскирует реальную проблему вместо ее решения
Наличие "фоновых задач" у каждого	Искусственно скрывает простой, не решая узкие места
Простой вместо помощи коллегам	Упускается возможность выравнивания загрузки и кросс-функциональности

Лабораторная работа 3
«Написание простого Unit-теста в парадигме TDD (JUnit/PyTest)»

1. Цель работы

Приобретение практических навыков:

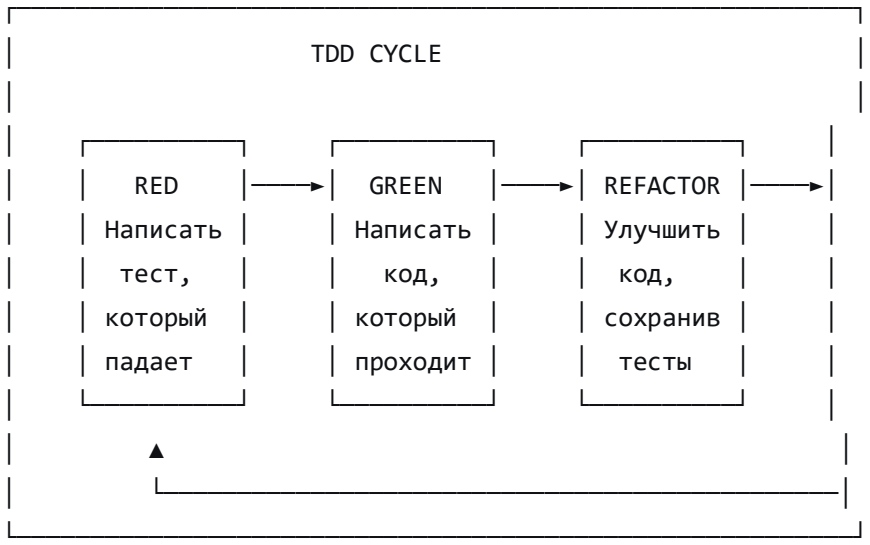
- Применения методологии Test-Driven Development (TDD) — разработки через тестирование;
- Написания unit-тестов с использованием фреймворков JUnit (Java) или PyTest (Python);
- Рефакторинга кода при сохранении прохождения тестов;
- Понимания цикла «Red — Green — Refactor».

2. Теоретическое введение

2.1. Что такое TDD?

Test-Driven Development (TDD) — это методология разработки программного обеспечения, при которой тесты пишутся до написания производственного кода. Цикл TDD состоит из трех этапов:

text



Этап	Описание
RED	Написать unit-тест на еще не существующую функциональность. Тест должен **упасть** (fail), так как код еще не написан
GREEN	Написать минимальный объем кода, достаточный для того, чтобы тест **прошел** (pass)
REFACTOR	Улучшить структуру кода, устранить дублирование, оптимизировать, не изменяя поведение (тесты должны оставаться зелеными)

2.2. Преимущества TDD

- **Гарантированная тестируемость** кода — код изначально проектируется с учетом тестирования;
- **Снижение количества багов** — регрессии обнаруживаются мгновенно;
- **Документация** — тесты служат живой документацией поведения системы;
- **Уверенность при рефакторинге** — можно смело изменять код;
- **Улучшение архитектуры** — TDD способствует слабой связанности компонентов.

2.3. Принципы хорошего unit-теста (FIRST)

Критерий	Описание
F — Fast	Быстрый (должен выполняться за доли секунды)
I — Isolated	Изолированный (не зависит от других тестов, не использует внешние ресурсы)
R — Repeatable	Повторяемый (дает одинаковый результат при каждом запуске)
S — Self-validating	Самопроверяемый (результат — pass/fail, без ручной проверки)
T — Timely	Своевременный (написан до или одновременно с производственным кодом)

2.4. Структура unit-теста (AAA Pattern)

```
text
// Arrange — подготовка данных
// Act     — выполнение тестируемого действия
// Assert  — проверка результата
```

2.5. Сравнение JUnit и PyTest

Характеристика	JUnit (Java)	PyTest (Python)
Аннотации/Маркеры	@Test, @BeforeEach	def test_*(), @pytest.fixture
Assertions	assertEquals(), assertTrue()	assert
Параметризация	@ParameterizedTest	@pytest.mark.parametrize
Фикстуры	@BeforeEach, @AfterEach	@pytest.fixture

3. Задание

3.1. Варианты заданий

Выберите один из двух вариантов в зависимости от предпочитаемого языка программирования.

Вариант A: Java (JUnit 5)

Контекст

Вы разрабатываете библиотеку для финансовых расчетов приложения «FitTrack» (трекер здоровья). Необходимо реализовать модуль расчета калорий.

Задание

Реализуйте класс CalorieCalculator с использованием TDD. Класс должен содержать следующие методы:

Метод	Описание
calculateBMR(weight, height, age, gender)	Расчет базового метаболизма (Basal Metabolic Rate) по формуле Миффлина — Сан-Жеора
calculateDailyCalories(bmr, activityLevel)	Расчет суточной нормы калорий с учетом уровня активности
calculateCaloriesBurned(activity,	Расчет калорий, сожженных при конкретной

Метод	Описание
duration, weight)	активности

Формулы:

BMR (мужчины): 88.362 + (13.397 × вес в кг) + (4.799 × рост в см) — (5.677 × возраст)

BMR (женщины): 447.593 + (9.247 × вес в кг) + (3.098 × рост в см) — (4.330 × возраст)

Уровни активности:

Уровень	Коэффициент
Минимальный (сидячий образ жизни)	1.2
Низкий (1-3 тренировки в неделю)	1.375
Средний (3-5 тренировок в неделю)	1.55
Высокий (6-7 тренировок в неделю)	1.725
Очень высокий (2 тренировки в день)	1.9

Вариант B: Python (PyTest)

Контекст

Вы разрабатываете модуль управления пользователями для приложения «FitTrack». Необходимо реализовать класс `UserManager` с использованием TDD.

Задание

Реализуйте класс `UserManager` с использованием TDD. Класс должен содержать следующие методы:

Метод	Описание

Метод	Описание
`register_user(username, email, password)`	Регистрация нового пользователя. Проверка уникальности username и email, сложности пароля
`authenticate_user(username, password)`	Аутентификация пользователя (возвращает токен или None)
`update_profile(username, **kwargs)`	Обновление данных профиля (email, пароль, метрики)
`get_user_by_username(username)`	Получение информации о пользователе
`delete_user(username)`	Удаление пользователя

Требования к паролю:

- Минимальная длина — 8 символов;
- Должен содержать хотя бы одну цифру;
- Должен содержать хотя бы одну заглавную букву;
- Должен содержать хотя бы один специальный символ (`!@#$%^&*`).

3.2. Этапы выполнения (для обоих вариантов)

Этап 1. Настройка окружения (5 баллов)

Для Java (JUnit 5):

1. Создайте проект Maven или Gradle.
2. Добавьте зависимость в `pom.xml`:

```xml
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.10.0</version>
    <scope>test</scope>
</dependency>
```

Для Python (PyTest):

1. Создайте виртуальное окружение: `python -m venv venv`
2. Установите PyTest: `pip install pytest`

3. Создайте структуру проекта:

```
text
project/
├── src/
│   └── user_manager.py      # или calorie_calculator.py
└── tests/
    └── test_user_manager.py  # или test_calorie_calculator.py
```

Этап 2. RED — Написание падающих тестов (15 баллов)

Напишите unit-тесты для **первого метода** (по одному из вариантов) до реализации самого метода.

Пример для Python (PyTest):

```python
# tests/test_user_manager.py
import pytest
from src.user_manager import UserManager

class TestUserManager:
    def test_register_user_success(self):
        """Тест успешной регистрации пользователя"""
        # Arrange
        manager = UserManager()

        # Act
        result = manager.register_user("john_doe", "john@example.com", "Pass123!")

        # Assert
        assert result is True
        assert "john_doe" in manager.get_all_users()

    def test_register_user_duplicate_username(self):
        """Тест регистрации с существующим username"""
        # Arrange
        manager = UserManager()
        manager.register_user("john_doe", "john@example.com", "Pass123!")

        # Act & Assert
        with pytest.raises(ValueError, match="Username already exists"):
            manager.register_user("john_doe", "jane@example.com", "Pass456!")
```

Пример для Java (JUnit):

```java
// src/test/java/CalorieCalculatorTest.java
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import static org.junit.jupiter.api.Assertions.*;

class CalorieCalculatorTest {

    private CalorieCalculator calculator;

    @BeforeEach
    void setUp() {
        calculator = new CalorieCalculator();
    }

    @Test
    void testCalculateBMR_Male() {
        // Arrange - данные подготовлены в setUp

        // Act
        double bmr = calculator.calculateBMR(80, 180, 30, "male");

        // Assert
        assertEquals(1832.0, bmr, 0.1);
    }

    @Test
    void testCalculateBMR_InvalidGender() {
        // Act & Assert
        assertThrows(IllegalArgumentException.class, () -> {
            calculator.calculateBMR(80, 180, 30, "invalid");
        });
    }
}
```

Требования к тестам:

- Напишите минимум **5 тестов** для первого метода;
- Покройте как позитивные (happy path), так и негативные сценарии;
- Используйте осмысленные названия тестов;
- Следуйте шаблону AAA (Arrange-Act-Assert).

Проверка: Запустите тесты — они должны **упасть** (RED).

Этап 3. GREEN — Минимальная реализация (15 баллов)

Напишите минимальный код, необходимый для прохождения всех написанных тестов.

Пример минимальной реализации (Python):

```python
# src/user_manager.py
class UserManager:
    def __init__(self):
        self._users = {}

    def register_user(self, username, email, password):
        if username in self._users:
            raise ValueError("Username already exists")

        # Минимальная валидация пароля
        if len(password) < 8:
            raise ValueError("Password too short")

        self._users[username] = {
            "username": username,
            "email": email,
            "password": password  # В реальном проекте — хешировать!
        }
        return True

    def get_all_users(self):
        return self._users.keys()
```

Требования:

- Не добавляйте лишней функциональности;
- Код должен проходить все тесты;
- Запустите тесты — теперь они должны **проходить** (GREEN).

Этап 4. REFACTOR — Улучшение кода (10 баллов)

1. Улучшите код, сохраняя зеленый статус тестов:
   - Вынесите валидацию пароля в отдельный метод;
   - Добавьте проверку формата email;
   - Улучшите обработку ошибок.
2. **Создайте тесты для второго метода** (например, `authenticate_user`), повторяя цикл RED-GREEN-REFACTOR.
3. Напишите минимум **3 теста** для второго метода.

Этап 5. Параметризованные тесты (10 баллов)

Реализуйте параметризованные тесты для проверки множества входных данных.

Пример для Python (PyTest):

python
```python
import pytest

@pytest.mark.parametrize("password,expected_valid", [
    ("Pass123!", True),      # корректный пароль
    ("pass123!", False),     # нет заглавной буквы
    ("PASS123!", False),     # нет строчной буквы
    ("Password!", False),    # нет цифры
    ("Pass123", False),      # нет спецсимвола
    ("123!", False),         # слишком короткий
])
def test_password_validation(password, expected_valid):
    manager = UserManager()
    if expected_valid:
        assert manager._validate_password(password) is True
    else:
        assert manager._validate_password(password) is False
```

Пример для Java (JUnit):

java
```java
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;

@ParameterizedTest
@CsvSource({
    "80, 180, 30, male, 1832.0",
    "70, 165, 25, female, 1488.0",
    "100, 190, 40, male, 2015.0"
})
void testCalculateBMR(double weight, double height, int age, String gender, double ex
pected) {
    double result = calculator.calculateBMR(weight, height, age, gender);
    assertEquals(expected, result, 0.1);
}
```

Этап 6. Интеграция с CI/CD (опционально, +5 баллов)

Настройте автоматический запуск тестов:

Для Java (Maven):

```
bash
mvn test
```

Для Python:

```
bash
pytest tests/ -v --cov=src
```

Создайте файл конфигурации для GitHub Actions:

```yaml
# .github/workflows/tests.yml
name: Run Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'
      - name: Install dependencies
        run: pip install pytest
      - name: Run tests
        run: pytest tests/ -v
```

4. Формат отчета

Отчет оформляется в виде текстового документа и должен содержать:

1. **Титульный лист** (название работы, ФИО, группа, выбранный вариант).
2. **Раздел 1. Описание выбранного варианта** и функциональности.
3. **Раздел 2. Листинг тестов (RED)** — код всех написанных тестов с комментариями.
4. **Раздел 3. Листинг минимальной реализации (GREEN)** — код, обеспечивающий прохождение тестов.
5. **Раздел 4. Листинг рефакторинга (REFACTOR)** — итоговый код после улучшений.
6. **Раздел 5. Параметризованные тесты** — код и результаты выполнения.
7. **Раздел 6. Скриншоты выполнения тестов**:

- Скриншот падающих тестов (RED);
- Скриншот проходящих тестов (GREEN);
- Скриншот покрытия кода тестами (если настроено).

8. **Раздел 7. Вывод** — анализ применения TDD, сложности, с которыми столкнулись, преимущества подхода.

5. Критерии оценки

Критерий	Максимальный балл
Правильная настройка окружения	5
Качество написанных тестов (RED) — покрытие сценариев	15
Корректность минимальной реализации (GREEN)	15
Качество рефакторинга (REFACTOR)	10
Параметризованные тесты	10
Оформление отчета, скриншоты	10
CI/CD настройка (опционально)	+5
Итого	**65**

6. Вопросы для защиты

1. В чем заключается цикл TDD? Опишите этапы RED-GREEN-REFACTOR.
2. Почему важно сначала написать падающий тест, а не сразу производственный код?
3. Чем отличается unit-тест от интеграционного теста?
4. Какие преимущества дает TDD при рефакторинге?
5. Как параметризованные тесты помогают в TDD?
6. Какие анти-паттерны в тестировании вы знаете? Приведите примеры.
7. В чем разница между `assertEquals` (JUnit) и `assert` (PyTest) с точки зрения читаемости?
8. Как TDD влияет на архитектуру приложения?

7. Дополнительные материалы

7.1. Распространенные ошибки при TDD

Ошибка	Исправление
Написание слишком сложного теста в RED	Разбейте на несколько простых тестов
Реализация кода без проверки на минимальность	GREEN должен быть минимальным — не пишите код, который не проверяется тестом
Пропуск этапа REFACTOR	Рефакторинг не менее важен, чем написание кода
Тестирование реализации, а не поведения	Тестируйте "что", а не "как"

7.2. Полезные команды

Python (PyTest):

```
bash
pytest tests/ -v                      # запуск всех тестов
pytest tests/test_user.py -v          # запуск конкретного файла
pytest tests/ -k "register"           # запуск тестов, содержащих "register"
pytest tests/ --cov=src               # оценка покрытия кода
pytest tests/ --cov=src --cov-report=html  # отчет в HTML
```

Java (Maven):

```
bash
mvn clean test                        # запуск всех тестов
mvn -Dtest=CalorieCalculatorTest test # запуск конкретного теста
mvn jacoco:report                     # отчет о покрытии
```

Лабораторная работа 4
«Настройка пайплайна CI/CD в GitLab CI»

1. Цель работы

Приобретение практических навыков:

- Создания и настройки CI/CD пайплайна в GitLab;
- Понимания структуры файла .gitlab-ci.yml и его ключевых элементов;
- Настройки стадий (stages) и заданий (jobs) пайплайна;
- Работы с артефактами (artifacts) и кэшированием (cache);
- Использования переменных окружения и предопределенных переменных GitLab CI.

2. Теоретическое введение

2.1. Что такое GitLab CI/CD?

GitLab CI/CD — это встроенная в GitLab система непрерывной интеграции и непрерывного развертывания, которая автоматизирует процессы сборки, тестирования и развертывания приложений .

Ключевые концепции :

Концепция	Описание
Pipeline (пайплайн)	Набор задач (jobs), объединенных в стадии (stages), которые выполняются в определенном порядке
Stage (стадия)	Группа jobs, которые выполняются параллельно. Стадии выполняются последовательно
Job (задание)	Минимальная единица выполнения в пайплайне, содержит скрипты для выполнения
Runner (раннер)	Агент, который выполняет задачи пайплайна
Artifacts (артефакты)	Файлы, создаваемые job'ом и передаваемые между стадиями
Variables (переменные)	Переменные окружения для конфигурации пайплайна

Концепция	Описание
Cache (кэш)	Механизм кэширования зависимостей для ускорения сборок

2.2. Файл .gitlab-ci.yml

Конфигурация CI/CD в GitLab задается с помощью YAML-файла с именем .gitlab-ci.yml, который помещается в корень репозитория .

Базовая структура :

```yaml
# Определение стадий пайплайна (порядок выполнения)
stages:
  - build
  - test
  - deploy

# Пример задания (job)
build-job:
  stage: build           # к какой стадии относится
  script:                # команды для выполнения
    - echo "Compiling the code..."
    - echo "Compile complete."

unit-test-job:
  stage: test
  script:
    - echo "Running unit tests..."
    - sleep 60
    - echo "Code coverage is 90%"

deploy-job:
  stage: deploy
  environment: production
  script:
    - echo "Deploying application..."
```

2.3. Стадии и задания

- **Стадии (**stages**)** определяют порядок выполнения групп заданий. Стадии выполняются последовательно, а задания внутри одной стадии — параллельно .
- Если стадии не определены явно, все задания по умолчанию попадают в стадию test .

2.4. Основные ключевые слова .gitlab-ci.yml

Ключевое слово	Описание
`stages`	Список стадий пайплайна
`stage`	Стадия для конкретного job'a
`script`	Команды для выполнения (обязательный параметр)
`before_script`	Команды, выполняемые перед основным скриптом
`after_script`	Команды, выполняемые после основного скрипта
`artifacts`	Файлы для сохранения и передачи между стадиями
`cache`	Кэширование файлов для ускорения последующих сборок
`variables`	Переменные окружения
`tags`	Метки для выбора runner'a
`allow_failure`	Разрешить failure задания (не останавливает пайплайн)
`when`	Условие выполнения (`on_success`, `on_failure`, `always`, `manual`)
`environment`	Окружение для деплоя (production, staging)
`dependencies`	Зависимости от артефактов других jobs
`needs`	Определение зависимостей между jobs для ускорения выполнения
`rules`	Условия для определения, когда создавать job

2.5. Предопределенные переменные GitLab CI

GitLab предоставляет множество предопределенных переменных :

Переменная	Описание
`CI_PIPELINE_ID`	Уникальный ID пайплайна
`CI_PIPELINE_SOURCE`	Источник запуска пайплайна (push, merge_request_event, schedule и др.)
`CI_COMMIT_BRANCH`	Название ветки коммита
`CI_COMMIT_SHA`	SHA коммита
`CI_PROJECT_DIR`	Путь к директории проекта
`CI_REGISTRY`	URL GitLab Container Registry
`CI_REGISTRY_IMAGE`	URL образа для текущего проекта в Container Registry
`GITLAB_USER_LOGIN`	Логин пользователя, запустившего пайплайн

3. Задание

3.1. Исходные данные

Вы — DevOps-инженер в команде разработки приложения **«FitTrack»** (знакомого по предыдущим лабораторным работам). Команда использует GitLab для управления кодом и хочет настроить автоматический CI/CD пайплайн.

Приложение: Простое веб-приложение на Node.js с Express.

Требования к пайплайну:

1. Автоматический запуск при каждом push в любую ветку;
2. Запуск при создании Merge Request;
3. Этапы пайплайна:
   - **build** — проверка синтаксиса и сборка;
   - **test** — запуск unit-тестов и линтинг кода;
   - **deploy** — развертывание (только для ветки main, вручную).

3.2. Этапы выполнения

Этап 1. Подготовка окружения (5 баллов)

1. **Создайте проект в GitLab:**
   - Зарегистрируйтесь/войдите в GitLab;
   - Создайте новый проект с названием `fitrack-cicd-lab`;
   - Установите видимость: Private.

2. **Создайте локальную копию репозитория:**

bash

```bash
git clone https://gitlab.com/your-username/fitrack-cicd-lab.git
cd fitrack-cicd-lab
```

3. **Создайте простое Node.js приложение:**

`package.json`:

json

```json
{
  "name": "fitrack-api",
  "version": "1.0.0",
  "description": "FitTrack API",
  "main": "app.js",
  "scripts": {
    "start": "node app.js",
    "test": "jest",
    "lint": "eslint ."
  },
  "dependencies": {
    "express": "^4.18.2"
  },
  "devDependencies": {
    "jest": "^29.7.0",
    "supertest": "^6.3.3",
    "eslint": "^8.56.0"
  }
}
```

`app.js`:

javascript

```javascript
const express = require('express');
const app = express();
const port = process.env.PORT || 3000;

app.use(express.json());

app.get('/health', (req, res) => {
  res.json({ status: 'ok', timestamp: new Date().toISOString() });
});
```

```
app.get('/api/users', (req, res) => {
  res.json({ users: [] });
});

if (require.main === module) {
  app.listen(port, () => {
    console.log(`FitTrack API listening on port ${port}`);
  });
}

module.exports = app;
```

`tests/app.test.js`:

```
javascript

const request = require('supertest');
const app = require('../app');

describe('FitTrack API', () => {
  test('GET /health returns 200 OK', async () => {
    const response = await request(app).get('/health');
    expect(response.statusCode).toBe(200);
    expect(response.body).toHaveProperty('status', 'ok');
  });

  test('GET /api/users returns 200 OK', async () => {
    const response = await request(app).get('/api/users');
    expect(response.statusCode).toBe(200);
    expect(response.body).toHaveProperty('users');
  });
});
```

4. **Добавьте файл `.gitignore`:**

```
text

node_modules/
.env
coverage/
.DS_Store
```

5. **Сделайте первый коммит и push:**

```
bash
```

```
git add .
git commit -m "Initial commit: FitTrack API"
git push origin main
```

Этап 2. Создание базового пайплайна (15 баллов)

Создайте файл `.gitlab-ci.yml` в корне репозитория с минимальной конфигурацией :

```yaml
# Определение стадий пайплайна
stages:
  - build
  - test
  - deploy

# Переменные окружения
variables:
  NODE_VERSION: "18"

# Базовые настройки для всех jobs
default:
  image: node:${NODE_VERSION}
  before_script:
    - npm install

# Job для стадии build
install-dependencies:
  stage: build
  script:
    - echo "Installing dependencies..."
    - npm ci
  artifacts:
    paths:
      - node_modules/
    expire_in: 1 hour

# Job для стадии test (линтер)
lint:
  stage: test
  script:
    - echo "Running linter..."
    - npm run lint || true  # временно не фейлим пайплайн
```

```yaml
# Job для стадии test (unit-тесты)
unit-test:
  stage: test
  script:
    - echo "Running unit tests..."
    - npm test
  artifacts:
    when: always
    paths:
      - coverage/
    reports:
      junit:
        - junit.xml

# Job для стадии deploy (только для main, вручную)
deploy-to-production:
  stage: deploy
  script:
    - echo "Deploying to production..."
    - echo "Application deployed successfully!"
  environment:
    name: production
    url: https://fitrack.example.com
  when: manual
  only:
    - main
```

Что нужно сделать:

1. Добавьте этот код в файл `.gitlab-ci.yml`;
2. Зафиксируйте изменения и отправьте в репозиторий;
3. Перейдите в GitLab → CI/CD → Pipelines и просмотрите результат выполнения.

Этап 3. Настройка кэширования (10 баллов)

Улучшите пайплайн, добавив кэширование зависимостей для ускорения последующих запусков :

Обновите `.gitlab-ci.yml`, добавив секцию `cache`:

```
yaml
default:
  image: node:${NODE_VERSION}
  cache:
```

```
    paths:
      - node_modules/
    key:
      files:
        - package-lock.json
  before_script:
    - npm ci --cache .npm --prefer-offline
```

Вопросы для анализа:

- Почему `node_modules/` добавляется в кэш?
- Какую роль играет `key: files: package-lock.json`?
- В чем разница между `cache` и `artifacts`?

Этап 4. Добавление линтинга и проверки кодстайла (10 баллов)

1. **Создайте конфигурацию ESLint** в файле `.eslintrc.json`:

   json

   ```
   {
     "env": {
       "node": true,
       "jest": true,
       "es2021": true
     },
     "extends": "eslint:recommended",
     "parserOptions": {
       "ecmaVersion": 2021
     },
     "rules": {
       "no-console": "warn",
       "no-unused-vars": "warn"
     }
   }
   ```

2. **Обновите** `package.json`, добавив скрипт линтинга:

   json

   ```
   "scripts": {
     "start": "node app.js",
     "test": "jest",
     "lint": "eslint .",
     "lint:fix": "eslint . --fix"
   ```

```
}
```

3. **Обновите job линтинга** в `.gitlab-ci.yml`:

```yaml
lint:
  stage: test
  script:
    - npm run lint
```

4. **Добавьте job проверки формата кода** (опционально):

```yaml
# Для Python можно использовать black, для JS можно добавить prettier
format-check:
  stage: test
  script:
    - npx prettier --check "**/*.js"
```

Этап 5. Настройка отчета о покрытии кода (10 баллов)

1. **Настройте Jest для генерации отчета о покрытии:**

`jest.config.js`**:**

```javascript
module.exports = {
  collectCoverage: true,
  coverageReporters: ['text', 'lcov', 'json-summary'],
  coverageDirectory: 'coverage',
  testMatch: ['**/tests/**/*.test.js'],
};
```

2. **Обновите job unit-тестов** для сохранения отчета о покрытии :

```yaml
unit-test:
  stage: test
  script:
    - npm test
  coverage: /All files[^|]*\|[^|]*\s+([\d\.]+)/
  artifacts:
    when: always
    paths:
```

```
      - coverage/
    reports:
      junit:
        - junit.xml
      coverage_report:
        coverage_format: cobertura
        path: coverage/cobertura-coverage.xml
```

3. **Настройте отображение покрытия кода в Merge Request** .

Этап 6. Настройка переменных окружения (10 баллов)

1. **Добавьте защищенные переменные в GitLab:**

- Перейдите в Settings → CI/CD → Variables;
- Добавьте переменную `DEPLOY_TOKEN` (защищенная, маскированная);
- Добавьте переменную `API_URL` = `https://api.fitrack.example.com`.

2. **Используйте переменные в пайплайне** :

```yaml
variables:
  NODE_VERSION: "18"
  # Переменная, определенная в пайплайне
  APP_NAME: "fitrack-api"

deploy-to-production:
  stage: deploy
  script:
    - echo "Deploying $APP_NAME to $API_URL"
    - echo "Using token: $DEPLOY_TOKEN"
    # В реальном проекте здесь был бы curl или скрипт деплоя
  environment:
    name: production
  when: manual
  only:
    - main
```

Этап 7. Настройка правил выполнения (rules) (10 баллов)

Обновите пайплайн с использованием `rules` для более гибкого управления выполнением jobs :

```yaml
# Правила для всей пайплайны (workflow)
```

```yaml
workflow:
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
    - if: $CI_COMMIT_BRANCH == "main"
    - if: $CI_COMMIT_BRANCH == "develop"
    - if: $CI_COMMIT_TAG
    - when: never

# Job с правилами выполнения
lint:
  stage: test
  script:
    - npm run lint
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
    - if: $CI_COMMIT_BRANCH == "main" || $CI_COMMIT_BRANCH == "develop"
    - when: never

unit-test:
  stage: test
  script:
    - npm test
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
    - if: $CI_COMMIT_BRANCH == "main" || $CI_COMMIT_BRANCH == "develop"
    - changes:
        - "app.js"
        - "tests/**/*"
        - "package*.json"
    - when: never
```

Этап 8. Работа с Merge Request пайплайнами (10 баллов)

1. **Создайте новую ветку** `feature/add-profile-endpoint`:

```bash
git checkout -b feature/add-profile-endpoint
```

2. **Добавьте новый эндпоинт в** `app.js`:

```javascript
app.get('/api/profile/:userId', (req, res) => {
```

```
  res.json({
    id: req.params.userId,
    name: 'John Doe',
    activityLevel: 'moderate'
  });
});
```

3. **Добавьте тесты в** `tests/app.test.js`:

```
javascript

test('GET /api/profile/:userId returns user profile', async () => {
  const response = await request(app).get('/api/profile/123');
  expect(response.statusCode).toBe(200);
  expect(response.body).toHaveProperty('id', '123');
  expect(response.body).toHaveProperty('name');
});
```

4. **Сделайте коммит и push**:

```
bash

git add .
git commit -m "feat: add profile endpoint"
git push origin feature/add-profile-endpoint
```

5. **Создайте Merge Request** через интерфейс GitLab и проследите за запуском пайплайна для MR.

Этап 9. Расширенная конфигурация (опционально, +10 баллов)

Добавьте в пайплайн один из следующих элементов:

Вариант A: Docker сборка

```
yaml
docker-build:
  stage: build
  image: docker:latest
  services:
    - docker:dind
  variables:
    DOCKER_HOST: tcp://docker:2375
    DOCKER_TLS_CERTDIR: ""
  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
```

```
    - docker build -t $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA .
    - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
  only:
    - main
```

Вариант B: Security сканирование (SAST)

```
yaml
include:
  - template: Security/SAST.gitlab-ci.yml
```

4. Формат отчета

Отчет оформляется в виде текстового документа и должен содержать:

1. **Титульный лист** (название работы, ФИО, группа).
2. **Раздел 1. Описание проекта** — краткое описание приложения FitTrack.
3. **Раздел 2. Файл .gitlab-ci.yml** — полный листинг с комментариями к каждому ключевому элементу.
4. **Раздел 3. Скриншоты пайплайнов**:
   - Скриншот успешного выполнения пайплайна в ветке main;
   - Скриншот выполнения пайплайна в Merge Request;
   - Скриншот раздела Pipelines с историей запусков.
5. **Раздел 4. Скриншоты логов jobs** — по одному примеру для каждой стадии.
6. **Раздел 5. Настройки проекта**:
   - Список переменных окружения (скриншот Settings → CI/CD → Variables);
   - Настройки раннеров (если настраивались).
7. **Раздел 6. Результаты Merge Request**:
   - Скриншот MR с проверками (pipelines, coverage);
   - Скриншот комментариев или статусов.
8. **Раздел 7. Ответы на контрольные вопросы**.
9. **Вывод** — анализ полученных навыков, сложности, с которыми столкнулись, предложения по улучшению.

5. Контрольные вопросы

1. Что такое GitLab Runner и какие типы раннеров существуют?
2. В чем разница между stages и jobs в GitLab CI/CD?
3. Как настроить кэширование зависимостей в пайплайне?
4. Что такое артефакты (artifacts) и как они используются между job'ами?
5. Как настроить различные окружения (staging, production)?

6. В чем преимущество Merge Request пайплайнов?
7. Как настроить scheduled пайплайны для ночных сборок?
8. Что такое `rules` и `only/except`? В чем разница?
9. Как использовать предопределенные переменные GitLab CI?
10. Какие best practices следует соблюдать при создании .gitlab-ci.yml?

6. Критерии оценки

Критерий	Максимальный балл
Подготовка окружения и создание проекта	5
Создание базового пайплайна (stages, jobs)	15
Настройка кэширования	10
Добавление линтинга и проверки кодстайла	10
Настройка отчета о покрытии кода	10
Использование переменных окружения	10
Настройка правил выполнения (rules)	10
Работа с Merge Request пайплайнами	10
Оформление отчета, скриншоты	10
Расширенная конфигурация (опционально)	+10
Итого	**90 (+10 опционально)**

7. Рекомендации по отладке

Частые проблемы и их решение

Проблема	Решение
Pipeline не запускается	Проверьте наличие файла `.gitlab-ci.yml` в корне репозитория. Используйте CI Lint (Build → Pipeline Editor → CI Lint)
Job fails с "command not found"	Проверьте, установлено ли необходимое ПО в Docker-образе. Установите зависимости в `before_script`
Artifacts не передаются	Проверьте `dependencies` и `needs` в конфигурации. Убедитесь, что source job выполнился успешно
Cache не работает	Проверьте `cache:key` — если изменился package-lock.json, cache будет инвалидирован
Docker build fails	Убедитесь, что используется service `docker:dind` и настроены переменные `DOCKER_HOST`

8. Дополнительные материалы

8.1. Полезные ссылки

- Официальная документация GitLab CI/CD
- CI/CD YAML Syntax Reference
- GitLab CI/CD Templates
- CI Lint Tool

8.2. Best practices

- Держите `.gitlab-ci.yml` в корне репозитория;
- Используйте `include` для разделения больших конфигураций ;
- Всегда указывайте `when: manual` для production-деплоя;

- Используйте `rules` вместо устаревших `only/except` ;
- Настраивайте `expire_in` для артефактов, чтобы экономить место .

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по организации и проведению самостоятельной работы по дисциплине
«Современные методологии инженерии программного обеспечения»
для студентов направления подготовки
09.04.04 Программная инженерия Направленность (профиль)
«Разработка, мониторинг и управление жизненным циклом
инженерных систем»

Ставрополь 2026

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 3
1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ 7
2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА 11
3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ 12
4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ 12
5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ 13

ВВЕДЕНИЕ

1. Цель и задачи освоения дисциплины (модуля)

Цель освоения дисциплины формирование у обучающихся системы теоретических знаний и практических навыков применения современных гибких (Agile) и масштабируемых методологий разработки программного обеспечения, а также развитие компетенций в области управления жизненным циклом сложных программных продуктов и организации эффективной командной работы..

Задачи освоения дисциплины:

- Изучить эволюцию методологий разработки ПО (от Waterfall до гибридных подходов).
- Освоить принципы, ценности и инструментарий гибких методологий (Scrum, Kanban, XP).
- Изучить подходы к масштабированию Agile для крупных организаций (SAFe, LeSS, Nexus).
- Сформировать навыки применения методологий управления требованиями и качеством в условиях неопределенности.
- Развить навыки командной работы, фасилитации и управления продуктом (Product Management).

2. Место дисциплины (модуля) в структуре образовательной программы

Дисциплина «Современные методологии инженерии программного обеспечения» относится к дисциплинам обязательной части.

3. Перечень планируемых результатов обучения по дисциплине (модулю), соотнесённых с планируемыми результатами освоения образовательной программы

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
УК-1. Способен осуществлять критический анализ проблемных ситуаций на основе системного подхода, вырабатывать стратегию действий	ИД-1$_{УК-1}$ осуществляет разработку стратегии действий для выявления и решения проблемной ситуации.	осуществляет разработку стратегии действий для выявления и решения проблемной ситуации.
	ИД-2 $_{УК-1}$ вырабатывает стратегию действий, принимает конкретные решения для ее реализации.	вырабатывает стратегию действий, принимает конкретные решения для ее реализации.
	ИД-3 $_{УК-1}$ ставит цели, определяет способы ее достижения, разрабатывает стратегий действий.	ставит цели, определяет способы ее достижения, разрабатывает стратегий действий.
ОПК-2. Способен разрабатывать оригинальные алгоритмы и программные средства, в том числе с использованием современных интеллектуальных технологий, для решения	ИД-1 $_{ОПК-2.}$ разрабатывать оригинальные алгоритмы для решения профессиональных задач;	Разрабатывает оригинальные алгоритмы для решения профессиональных задач;
	ИД-2 $_{ОПК-2.}$ разрабатывать программные средства для решения профессиональных задач	Разрабатывает программные средства для решения профессиональных задач
	ИД-3 $_{ОПК-2.}$ использовать современные интеллектуальные технологии для решения	Использует современные интеллектуальные технологии для решения

профессиональных задач	профессиональных задач	профессиональных задач

4. Объем учебной дисциплины и формы контроля *

Объем занятий: всего: 4 з.е. 144 ак.ч.	ОФО, в астр. часах
Контактная работа:	32
Лекции/из них практическая подготовка	16
Лабораторных работ/из них практическая подготовка	16
Практических занятий/из них практическая подготовка	
Самостоятельная работа	112
Формы контроля	
Экзамен	
Зачет	
Зачет с оценкой	
Расчетно-графические работы	
Курсовые работа	
Контрольные работы	

* Дисциплина (модуль) предусматривает применение электронного обучения, дистанционных образовательных технологий

5. Содержание дисциплины (модуля), структурированное по темам (разделам) с указанием количества часов и видов занятий

№	Раздел (тема) дисциплины и краткое содержание	Формируемые компетенции, индикаторы	очная форма			
			Контактная работа обучающихся с преподавателем /из них в форме практической подготовки, часов			Самостоятельная работа, часов
			Лекции	Практические занятия	Лабораторные работы	
1	Введение. Эволюция методологий 1. Классические методологии (Waterfall, V-Model). 2. Причины возникновения гибких методологий. 3. Манифест Agile: ценности и принципы. 4. Обзор современных подходов (Agile, Lean, DevOps).	УК-1; ОПК-2	2		-	12
2	Гибкие методологии (Agile). Scrum 1. Роли в Scrum: Product Owner, Scrum Master, Development Team. 2. Артефакты: Product Backlog, Sprint Backlog, Increment. 3. События: Sprint, Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective. 4. Управление требованиями через User Stories и Definition of Done (DoD).	УК-1; ОПК-2	4		4	14

3	Kanban и Lean. XP (Extreme Programming) 1. Принципы Kanban: визуализация потока (Jira/Trello), ограничение WIP (Work in Progress), управление потоком. 2. Lean Software Development: устранение потерь (Muda). 3. Практики XP: парное программирование, TDD (Test-Driven Development), непрерывная интеграция, коллективное владение кодом.	УК-1; ОПК-2	2		4	14
4	Масштабирование Agile (SAFe, LeSS) 1. Проблемы масштабирования (Scrum of Scrums). 2. Обзор фреймворков: SAFe (Scaled Agile Framework): уровни (Team, Program, Large Solution, Portfolio); LeSS (Large Scale Scrum): принципы и структура. 3. Nexus. Синхронизация релизов и планирование на уровне портфеля.	УК-1; ОПК-2	4		4	14
5	DevOps культура и Continuous Delivery 1. Интеграция разработки и эксплуатации. 2. CI/CD (Continuous Integration / Continuous Delivery). 3. Инструментарий: Git, Jenkins/GitLab CI, Docker, Kubernetes. 4. Управление инфраструктурой как кодом (IaC). 5. Метрики: DORA metrics (Lead Time, Deployment Frequency, MTTR, Change Failure Rate).	УК-1; ОПК-2	2		2	14

6	Управление продуктом и стейкхолдерами 1. Product Management vs Project Management. 2. Определение ценности продукта (MVP — Minimum Viable Product). 3. Управление ожиданиями заказчика. 4. Работа с командами: мотивация, фасилитация ретроспектив, разрешение конфликтов. 5. Техники оценки трудоемкости (Story Points, Planning Poker).	УК-1; ОПК-2	2		2	14
	ИТОГО за 2 семестр		16		16	112
	ИТОГО		16		16	112

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Все виды самостоятельной работы по дисциплине " Анализ данных в эксплуатации программных систем " приведены в таблице «Технологическая карта самостоятельной работы». В основу самостоятельной работы студентов по дисциплине положено конспектирование лекций и рекомендуемых источников, подготовка к компьютерному тестированию знаний, оформление и подготовка к защите отчетов по лабораторным работам, выполнению разноуровневых индивидуальных заданий в ходе лабораторных работ. По каждому виду самостоятельной работы предусмотрены определённые формы отчетности, которые студенты предъявляют преподавателю в ходе лабораторных занятий или в часы индивидуальных консультаций.

Таблица 1 - Технологическая карта самостоятельной работы.

Код оцениваемой компетенции, индикаторов	Этап формирования компетенции (№ темы) (в соответствии с рабочей программой дисциплины)	Средства и технологии оценки	Вид контроля, аттестация (текущий/промежуточный)	Тип контроля (устный, письменный или с использованием технических средств)	Наименование оценочного средства
1 семестр					
УК-1.	Темы № 1–3	Собеседование, оценка ответов по вопросам заданных тем	текущий	устный	вопросы для собеседования
ОПК-2.	Темы № 4–6	Собеседование, оценка ответов по вопросам заданных тем	текущий	устный	вопросы для собеседования

Для успешного освоения дисциплины, необходимо выполнить указанные в таблице 1 виды самостоятельной работы, используя рекомендуемые источники информации.

8.1. Перечень основной и дополнительной литературы, необходимой для освоения дисциплины (модуля)

8.1.1. Перечень основной литературы:

1. Нехорошкова, Л. Г. Информационное моделирование и анализ требований : учебное пособие : [16+] / Л. Г. Нехорошкова ; Поволжский государственный технологический университет. – Йошкар-Ола : Поволжский государственный технологический университет, 2020. – 146 с. : ил., табл., схем. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=615678 (дата обращения: 30.03.2026). – Библиогр.: с. 113-114. – ISBN 978-5-8158-2209-2. – Текст : электронный.

2. Лисяк, В. В. Разработка информационных систем : учебное пособие : [16+] / В. В. Лисяк ; Южный федеральный университет. – Ростов-на-Дону ; Таганрог : Южный федеральный университет, 2019. – 97 с. : ил. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=577875 (дата обращения: 30.03.2026). – Библиогр.: с. 91 - 93. – ISBN 978-5-9275-3168-4. – Текст : электронный.

3. Гибкая методология разработки программного обеспечения : учебное пособие : [16+] / Национальный открытый университет «ИНТУИТ». – Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2010. – 134 с. : ил. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=233769 (дата обращения: 30.03.2026). – Текст : электронный.

8.1.2. Перечень дополнительной литературы:

1. Волина, С. А. Учебник немецкого языка для первого года обучения языкового вуза : учебник для вуза / С. А. Волина, Г. Б. Воронина, Л. М. Карпова. - М. : ИЛБИ, 1997. - 461 с. : ил. - ISBN 5-87483-047-2

2. Зайцева, Л.В. Иностранный язык: english for nature managers / Л.В. Зайцева. – Воронеж : Воронежская государственная лесотехническая академия, 2013. – 176 с. – Режим доступа: по подписке. – URL: http://biblioclub.ru/index.php?page=book&id=142300 (дата обращения: 04.03.2026). – ISBN 978-5-7994-0537-3. – Текст : электронный.

8.2. Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине

1. Управление проектами : фундаментальный курс : учебник : [16+] / А. В. Алешин, В. М. Аньшин, К. А. Багратиони [и др.] ; под ред. В. М. Аньшина, О. Н. Ильиной. – Москва : Издательский дом Высшей школы экономики, 2022. – 800 с. : ил., табл. – (Учебники Высшей школы экономики). – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=699578 (дата обращения: 30.03.2026). – Библиогр. в кн. – ISBN 978-5-7598-2313-1 (в пер.). – ISBN 978-5-7598-2413-8 (e-book). – DOI 10.17323/978-5-7598-2313-1. – Текст : электронный.

2. Преображенская, Т. В. Управление проектами : учебное пособие : [16+] / Т. В. Преображенская, М. Ш. Муртазина, А. А. Алетдинова ; Новосибирский государственный технический университет. – Новосибирск : Новосибирский государственный технический университет, 2018. – 123 с. : ил., табл. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=574957 (дата обращения: 30.03.2026). – Библиогр. в кн. – ISBN 978-5-7782-3558-8. – Текст : электронный.

8.3. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

www.biblioclub.ru - Электронная библиотечная система «Университетская библиотека онлайн»

www.eLibrary.ru - Научная электронная библиотека

www.iprbookshop.ru - Электронно-библиотечная система IPRbooks

Особенности освоения дисциплины (модуля) лицами с ограниченными возможностями здоровья

Обучающимся с ограниченными возможностями здоровья предоставляются специальные учебники, учебные пособия и дидактические материалы, специальные технические средства обучения коллективного и индивидуального пользования, услуги ассистента (помощника),

оказывающего

обучающимся необходимую техническую помощь, а также услуги сурдопереводчиков и тифлосурдопереводчиков.

Освоение дисциплины (модуля) обучающимися с ограниченными возможностями здоровья может быть организовано совместно с другими обучающимися, а также в отдельных группах.

Освоение дисциплины (модуля) обучающимися с ограниченными возможностями здоровья осуществляется с учетом особенностей психофизического развития, индивидуальных возможностей и состояния здоровья.

В целях доступности получения высшего образования по образовательной программе лицами с ограниченными возможностями здоровья при освоении дисциплины (модуля) обеспечивается:

1) для лиц с ограниченными возможностями здоровья по зрению:

- присутствие ассистента, оказывающий студенту необходимую техническую помощь с учетом индивидуальных особенностей (помогает занять рабочее место, передвигаться, прочитать и оформить задание, в том числе, записывая под диктовку),
- письменные задания, а также инструкции о порядке их выполнения оформляются увеличенным шрифтом,
- специальные учебники, учебные пособия и дидактические материалы (имеющие крупный шрифт или аудиофайлы),
- индивидуальное равномерное освещение не менее 300 люкс,
- при необходимости студенту для выполнения задания предоставляется увеличивающее устройство;

2) для лиц с ограниченными возможностями здоровья по слуху:

- присутствие ассистента, оказывающий студенту необходимую техническую помощь с учетом индивидуальных особенностей (помогает занять рабочее место, передвигаться, прочитать и оформить задание, в том числе, записывая под диктовку),
- обеспечивается наличие звукоусиливающей аппаратуры коллективного пользования, при необходимости обучающемуся предоставляется звукоусиливающая аппаратура индивидуального пользования;
- обеспечивается надлежащими звуковыми средствами воспроизведения информации;

3) для лиц с ограниченными возможностями здоровья, имеющих нарушения опорно-двигательного аппарата (в том числе с тяжелыми нарушениями двигательных функций верхних конечностей или отсутствием верхних конечностей):

- письменные задания выполняются на компьютере со специализированным программным обеспечением или надиктовываются ассистенту;
- по желанию студента задания могут выполняться в устной форме.

Особенности реализации дисциплины с применением дистанционных образовательных технологий и электронного обучения

Согласно части 1 статьи 16 Федерального закона от 29 декабря 2012 г. № 273-ФЗ «Об образовании в Российской Федерации» под *электронным обучением* понимается организация образовательной деятельности с применением содержащейся в базах данных и используемой при реализации образовательных программ информации и обеспечивающих ее обработку информационных технологий, технических средств, а также информационно-телекоммуникационных сетей, обеспечивающих передачу по линиям связи указанной информации, взаимодействие обучающихся и педагогических работников. Под *дистанционными образовательными технологиями* понимаются образовательные технологии, реализуемые в основном с применением информационно- телекоммуникационных сетей при опосредованном (на расстоянии) взаимодействии обучающихся и педагогических работников.

Реализация дисциплины может быть осуществлена с применением дистанционных образовательных технологий и электронного обучения полностью или частично. Компоненты УМК дисциплины (рабочая программа дисциплины, оценочные и методические материалы, формы аттестации), реализуемой с применением дистанционных образовательных технологий и электронного обучения, содержат указание на их использование.

При организации образовательной деятельности с применением дистанционных

образовательных технологий и электронного обучения могут предусматриваться асинхронный и синхронный способы осуществления взаимодействия участников образовательных отношений посредством информационно-телекоммуникационной сети «Интернет».

При применении дистанционных образовательных технологий и электронного обучения в

расписании по дисциплине указываются: способы осуществления взаимодействия участников образовательных отношений посредством информационно-телекоммуникационной сети «Интернет» (ВКС-видеоконференцсвязь, ЭТ – электронное тестирование); ссылки на электронную информационно- образовательную среду СКФУ, на образовательные платформы и ресурсы иных организаций, к которым предоставляется открытый доступ через информационно-телекоммуникационную сеть

«Интернет»; для синхронного обучения - время проведения онлайн-занятий и преподаватели; для асинхронного обучения - авторы онлайн-курсов.

При организации промежуточной аттестации с применением дистанционных образовательных технологий и электронного обучения используются Методические рекомендации по применению технических средств, обеспечивающих объективность результатов при проведении промежуточной и государственной итоговой аттестации по образовательным программам высшего образования - программам бакалавриата, программам специалитета и программам магистратуры с применением дистанционных образовательных технологий (Письмо Минобрнаки России от 07.12.2020 г. № МН- 19/1573-АН "О направлении методических рекомендаций").

Реализация дисциплины с применением электронного обучения и дистанционных образовательных технологий осуществляется с использованием электронной информационно-образовательной среды СКФУ, к которой обеспечен доступ обучающихся через информационно-телекоммуникационную сеть «Интернет», или с использованием ресурсов иных организаций, в том числе платформ, предоставляющих сервисы для проведения видеоконференций, онлайн-встреч и дистанционного обучения (МТС-Линк), а также с использованием возможностей социальных сетей для осуществления коммуникации обучающихся и преподавателей.

Учебно-методическое обеспечение дисциплины, реализуемой с применением электронного обучения и дистанционных образовательных технологий, включает представленные в электронном виде рабочую программу, учебно-методические пособия или курс лекций, методические указания к выполнению различных видов учебной деятельности обучающихся, предусмотренных дисциплиной, и прочие учебно-методические материалы, размещенные в информационно-образовательной среде СКФУ.

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

При самостоятельном изучении тем лекционных занятий и заданной литературы студентам рекомендуется:

- в день проведения занятий проработать, изученный на занятии учебный материал по конспекту. При этом необходимо выделить вопросы, на которые следует обратить большее внимание при дальнейшем изучении текущей темы. После проработки учебного материала необходимо ответить на, рекомендуемые контрольные вопросы;

-с целью качественного закрепления изученного материала в последующем следует более детально проработать учебный материал с использованием рекомендованной литературы. Студентам рекомендуется выделить вопросы, требующие более детальной проработки с помощью преподавателя;

- накануне проведения лекции студенту рекомендуется закрепить ранее изученный теоретический материал, подготовить вопросы, требующие уточнения у преподавателя.

Рекомендации по организации работы с литературой

Для овладения, закрепления и систематизации знаний студентам рекомендуется

самостоятельная работа с рекомендованной литературой и конспектом лекций. При самостоятельной работе с рекомендованной литературой студентам рекомендуется проводить конспектирование учебного материала, изложенного в рекомендованных источниках.

Конспектирование источников проводится при детальном изучении темы, при этом студентам рекомендуется:

- дополнять конспект лекционного занятия путем его расширения по наиболее важным вопросам, рассмотренным на занятии. Конспектирование источников заключается в дополнении конспекта следующими положениями:

-основные определения и их физический смысл;

- основные математические соотношения с раскрытием их физического смысла;
- выводы по учебным вопросам, изучаемой темы занятий.

Контроль этого вида самостоятельной работы осуществляется на учебных занятиях путем проверки преподавателем конспекта лекций и собеседования.

При конспектировании источников студенты должны исходить из рационального объема, конспектируемого материала. Конспект должен быть кратким и понятным при его использовании после изучения текущей темы.

При конспектировании источников рекомендуется отдельные рисунки, наиболее важные их фрагменты, наиболее важные формулы выделять цветом. При ведении конспекта на занятии рекомендуется заполнять две трети страницы листа тетради по вертикали, а свободную часть использовать при дополнении конспекта, по изучаемому вопросу.

Для успешного освоения дисциплины, необходимо самостоятельно детально изучить представленные темы по рекомендуемым источникам информации.

Конспектирование источников оценивается удовлетворительно и неудовлетворительно. Работа по конспектированию источников оценивается удовлетворительно если:

- конспект имеет достаточный объем детализации по, изучаемой теме, обеспечивающий заданный уровень освоения теоретического материала дисциплины;
- конспект оформлен аккуратно, изучаемый материал изложен последовательно в соответствии с порядком изучения дисциплины, содержит обобщающие выводы по каждой теме.

Работа по конспектированию источников оценивается неудовлетворительно, если не выполнены требования на оценку удовлетворительно.

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

При выполнении практикума по дисциплине для достижения базового уровня сформированности компетенций студент должен уметь применять базовые знания по разделам дисциплины, владеть навыками самостоятельного решения учебных задач базового уровня.

При выполнении практикума по дисциплине для достижения повышенного уровня сформированности компетенций студент должен уметь применять повышенные знания по разделам дисциплины, владеть навыками самостоятельного решения учебных задач повышенного уровня.

Защита практической работы осуществляется путем собеседования студента с преподавателем. При собеседовании студент представляет на проверку отчет по практической работе. Собеседование со студентом, претендующим на оценку «отлично» проводится по вопросам и практическим заданиям повышенного уровня обучения. Собеседование со студентом, не претендующим на оценку «отлично» проводится по

вопросам и практическим заданиям базового уровня обучения.

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ

ПО ДИСЦИПЛИНЕ Управление жизненным циклом инженерных систем

Базовый уровень

1. Классические методологии (Waterfall, V-Model).
2. Причины возникновения гибких методологий.
3. Роли в Scrum: Product Owner, Scrum Master, Development Team.
4. Артефакты: Product Backlog, Sprint Backlog, Increment.
5. Принципы Kanban: визуализация потока (Jira/Trello), ограничение WIP (Work in Progress), управление потоком.
6. Lean Software Development: устранение потерь (Muda).
7. Проблемы масштабирования (Scrum of Scrums).
8. Обзор фреймворков: SAFe (Scaled Agile Framework): уровни (Team, Program, Large Solution, Portfolio); LeSS (Large Scale Scrum): принципы и структура.
9. Интеграция разработки и эксплуатации.
10. CI/CD (Continuous Integration / Continuous Delivery).
11. Product Management vs Project Management.
12. Определение ценности продукта (MVP — Minimum Viable Product).

Повышенный уровень

1. Манифест Agile: ценности и принципы.
2. Обзор современных подходов (Agile, Lean, DevOps).
3. События: Sprint, Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective.
4. Управление требованиями через User Stories и Definition of Done (DoD).
5. Практики XP: парное программирование, TDD (Test-Driven Development), непрерывная интеграция, коллективное владение кодом.
6. Nexus. Синхронизация релизов и планирование на уровне портфеля.
7. Инструментарий: Git, Jenkins/GitLab CI, Docker, Kubernetes.
8. Управление инфраструктурой как кодом (IaC).
9. Метрики: DORA metrics (Lead Time, Deployment Frequency, MTTR, Change Failure Rate).
10. Управление ожиданиями заказчика.
11. Работа с командами: мотивация, фасилитация ретроспектив, разрешение конфликтов.
12. Техники оценки трудоемкости (Story Points, Planning Poker).

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

Оценка «отлично» выставляется студенту, если он глубоко и прочно усвоил программный материал, исчерпывающе, последовательно, четко и логически стройно его излагает, умеет тесно увязывать теорию с практикой, свободно справляется с вопросами не только базового, но и повышенного уровня, причем не затрудняется с ответом при видоизменении заданий, правильно обосновывает принятое решение.

Оценка «хорошо» выставляется студенту, если он твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопросы базового уровня, правильно применяет теоретические положения при решении практических вопросов базового уровня, владеет необходимыми навыками и приемами их выполнения.

Оценка «удовлетворительно» выставляется студенту, если он имеет знания только основного материала базового уровня, но не усвоил его деталей, допускает неточности, недостаточно правильные формулировки, нарушения логической последовательности в изложении программного материала.

Оценка «неудовлетворительно» выставляется студенту, не знающему значительной части программного материала, допускающему грубые ошибки.

Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций

Процедура проведения данного оценочного мероприятия включает в себя собеседование.

Предлагаемые студенту задания позволяют проверить компетенции УК-1; ОПК-2. При проверке задания оценивается:

– объем и глубина усвоенного программного материала;

–последовательность, четкость и логическая стройность его изложения.