МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ
СОВРЕМЕННЫЙ МЕЖДУНАРОДНЫЙ УНИВЕРСИТЕТ

КАФЕДРА ИНФОРМАТИКИ

«Согласовано»
Начальник учебной части
_____Ерке Е.Э.
«25»__08__2024г.

«Утверждаю»
I-проректор СМУ
_____Максат Макамбай
«_____»_____20___г.

РАБОЧАЯ ПРОГРАММА
ПРОГРАММНАЯ ИНЖЕНЕРИЯ

для специальности (направления специальности): **710300 «Прикладная информатика»**
Форма обучения: очная/заочная
Виды учебной деятельности:

Программа	Всего часов				СРС	СРС II	Форма отчетности
	всего	аудиторных	лекция	Прак.раб			
3-курс, 6 семестр (о\о)	5 кр 150 час	75	30	45	45 ч	30 ч	экзамен
4-курс, 7 семестр (з\о)	5 кр 150 час	40	10	30	80 ч	30 ч	экзамен

«Согласовано»
Профилирующей кафедрой
_____зав.каф.
от «_____»_____20___г.

«Рассмотрено»
на заседании кафедры
_____
от «_____»_____20__ г.

Рабочая программа дисциплины разработана в соответствии с ГОС (1578/1 от 21 сентября 2021 г), ОПОП направления 710300 - Прикладная информатика, Положения о разработке УМК в СМУ.

Разработчик: Абилов К.Б.

Жалал-Абад
2024 г.

Содержание

1.1 Реквизиты дисциплины

Дисциплина : «Программная инженерия»

Курс: 3

Направление: 710300 – Прикладная информатика

Количество кредит часов: 5 кр (150 ч)

Форма обучения: очная/заочная

1.2 Объем дисциплины и виды учебной работы

Вид учебной работы	Семестр 6 о/о Всего часов 150 ч	Семестр 7 з/о Всего часов 150 ч
Аудиторные занятия (всего)	**75**	**40**
В том числе:		
Лекции (Л)	30	10
Практические занятия (ПЗ)	45	30
Семинары (С)		
Лабораторные практикумы (ЛП)		
Клинические практические занятия (КПЗ)		
Самостоятельная работа (всего)	**75**	**110**
СРСП	30	30
СРС	45	80
Форма контроля	экзамен	экзамен
Общая трудоемкость (час.)	**150 ч**	**150 ч**

2. Место дисциплины в структуре ООП

Учебная дисциплина «Программная инженерия» является специальной дисциплиной, формирующей профессиональные знания, необходимые для будущей трудовой деятельности.

Дисциплина относится к базовой части профессионального цикла.

3. Пререквизиты и постреквизиты курса

Для успешного освоения курса «Программная инженерия» студент должен обладать базовыми знаниями и навыками, полученными в рамках следующих дисциплин:

1. **Информатика** – знание основ алгоритмизации, структур данных, принципов программирования.

2. **Языки и методы программирования** – умение писать и отлаживать код на одном из современных языков (Python, Java, C++ и др.).
3. **Алгоритмы и структуры данных** – понимание логики и оптимальности программных решений.
4. **Базы данных** – знание принципов построения и управления реляционными СУБД.
5. **Операционные системы** – знание архитектуры ОС, процессов, потоков, управления памятью.
6. **Математическая логика и дискретная математика** – логическое мышление, понимание формальных моделей и схем.
7. **Теория систем и системный анализ** – основы анализа требований и структурного подхода к проектированию.

Постреквизиты (дисциплины, использующие знания курса):
После освоения курса «Программная инженерия» студенты смогут успешно изучать и применять знания в следующих курсах и модулях:
1. **Управление проектами в ИТ** – применение инженерных принципов в планировании, реализации и контроле ИТ-проектов.
2. **Разработка программных продуктов и систем** – создание промышленного ПО с использованием жизненного цикла разработки.
3. **Архитектура программного обеспечения** – построение устойчивых и масштабируемых архитектур.
4. **Тестирование и обеспечение качества ПО** – формирование подходов к верификации, тест-дизайну и автоматизации тестирования.
5. **DevOps и CI/CD практики** – применение инженерных подходов к автоматизации разработки и доставки ПО.
6. **Проектный практикум / Курсовое и дипломное проектирование** – реализация собственных программных продуктов от идеи до реализации.
7. **Информационная безопасность ПО** – защита архитектур и кода от уязвимостей.
8. **Инженерия требований и UX-дизайн** – построение ориентированных на пользователя систем на основе анализа требований.

4. Необходимость изучения курса

Современное общество стремительно цифровизуется, и программное обеспечение стало ключевым компонентом практически

всех сфер человеческой деятельности — от экономики и образования до здравоохранения и безопасности. В этой связи возникает необходимость в системном подходе к проектированию, разработке, тестированию и сопровождению программных продуктов.

Курс **«Программная инженерия»** обеспечивает студентов знаниями, навыками и методами, необходимыми для эффективной организации полного жизненного цикла программного обеспечения — от анализа требований до сопровождения и модернизации готовых систем.

Изучение данной дисциплины позволяет:

- понимать принципы командной разработки и распределения ролей в ИТ-проектах;
- осваивать методы управления качеством, рисками и конфигурацией программных продуктов;
- работать с инженерной документацией и требованиями заказчика;
- применять современные инструменты и технологии (системы контроля версий, CI/CD, UML, трекеры задач и др.);
- закладывать основу для проектной, исследовательской и стартап-деятельности в ИТ-сфере.

Таким образом, курс является **базовой дисциплиной** профессионального цикла, необходимой для подготовки квалифицированных разработчиков, архитекторов и менеджеров программных систем.

5. Цели и задачи курса

Цель курса:

Формирование у студентов целостного представления о принципах, методах и инструментах программной инженерии, обеспечивающих эффективную разработку, сопровождение и управление программными продуктами в соответствии с требованиями заказчика, стандартами качества и ограничениями ресурсов.

Задачи курса:

1. **Познакомить** студентов с основами жизненного цикла программного обеспечения: анализ требований, проектирование, кодирование, тестирование, внедрение и сопровождение.

2. **Научить** применять методы и стандарты инженерии ПО, включая каскадную, итеративную, гибкую (Agile, Scrum, Kanban) модели разработки.
3. **Сформировать навыки** работы с инструментами коллективной разработки: системы контроля версий, трекеры задач, CI/CD-платформы.
4. **Развить умения** анализа требований и построения проектной документации (техническое задание, спецификации, UML-диаграммы).
5. **Обучить методам** управления качеством, рисками, затратами и временными ресурсами в программных проектах.
6. **Формировать культуру** командной разработки, эффективной коммуникации, распределения ролей и ответственности.
7. **Подготовить студентов** к применению инженерного подхода в разработке собственных программных решений, стартапов и дипломных проектов.

6. Компетенции обучающегося, формируемые в результате освоения дисциплины

Формирование компетенций для направления: 710300 «Прикладная информатика»

Код компетенции	Компетенция	Знания	Умения	Навыки
ПК-7	Способен обосновывать выбор алгоритмов, ПО и ОС при проектировании ИС, программировать и тестировать приложения	– Основы алгоритмизации и теории вычислений– Типы и свойства ОС, языков программирования, СУБД	– Выбирать адекватные алгоритмы обработки данных– Программировать и проводить модульное тестирование	– Реализация программ на выбранном языке (Python, Java и др.)– Отладка и верификация программного кода
ПК-14	Способен использовать технологические и функциональные стандарты, современные	– Стандарты качества ПО (ISO/IEC 25010, ГОСТ)– Методы оценки надёжности, отказоустойчиво	– Применять модели оценки качества на практике– Использовать стандарты и документацию	– Использование средств статического и динамического анализа–

Код компетенции	Компетенция	Знания	Умения	Навыки
	модели и методы оценки качества и надежности при проектировании, конструировании и отладке программных средств	сти и устойчивости ПО	при проектировании	Тест-дизайн, юнит- и нагрузочное тестирование, баг-трекинг
ПК-15	Способен анализировать и выбирать методы и средства обеспечения информационной безопасности	– Основы информационной безопасности и киберугроз– Методы криптографии, аутентификации, разграничения доступа	– Анализировать риски и уязвимости в ПО и инфраструктуре– Подбирать эффективные методы защиты информации	– Реализация базовых мер защиты (SSL, хеширование, роли доступа)– Настройка средств защиты: фаерволы, антивирусы, мониторинг активности

В результате освоения учебной дисциплины обучающийся должен ЗНАТЬ:

• основные этапы жизненного цикла программного обеспечения (SDLC), их назначение и взаимосвязь;

• модели и методы проектирования программных систем (каскадная, спиральная, итеративная, Agile и др.);

• принципы и стандарты разработки программных продуктов (ISO/IEC, IEEE, ГОСТ);

• методы документирования требований, проектной и пользовательской документации;

• подходы к тестированию, верификации и валидации ПО;

• принципы управления проектами в программной инженерии, включая оценку рисков, затрат и качества;

• инструменты командной разработки, контроля версий, CI/CD и DevOps-практики.

УМЕТЬ:

• анализировать требования заказчика и формализовывать их в техническое задание;

• выбирать и обосновывать архитектурные решения при проектировании программных систем;

• разрабатывать компоненты программного обеспечения с использованием современных языков программирования;

• применять методы модульного и интеграционного тестирования;

• использовать инструменты управления версиями и задачами (Git, Jira, Trello и др.);

• оформлять техническую документацию, спецификации, отчёты по качеству.

ВЛАДЕТЬ:

• навыками командной работы в условиях реального проекта;

• методами и инструментами организации полного жизненного цикла ПО;

• технологиями непрерывной интеграции и доставки программных продуктов (CI/CD);

• средствами моделирования архитектурных решений (UML-диаграммы, диаграммы потоков и состояний);

• навыками обеспечения качества, отказоустойчивости и безопасности программных систем.

7. Образовательные технологии

Изучение дисциплины предполагает использование традиционных способов коллективного обучения – *лекций, лабораторных занятий, индивидуальных заданий с последующей отчетностью.*

Применяемые информационные технологии: *лекции в форме презентаций, обучающие и тестирующие программы, электронные учебники.*

Оценочные средства для текущего контроля успеваемости, рубежного контроля по итогам освоения дисциплины и учебно-методическое обеспечение самостоятельной работы бакалавров:

- **формой текущего контроля знаний студентов (аудиторные занятия)** *является контроль посещения лекционных и практических занятий, активность студентов на практических*

занятиях. Каждый из двух текущих контролей оценивается по ***30 баллов.***

формой текущего контроля знаний студентов (внеаудиторные занятия) *является контроль СРСП, участие в НИРС (выступление на студенческой конференции, публикация статей)*

- **формой итогового контроля знаний и умений** бакалавров **по курсу** *является экзамен.*

Текущий и рубежный контроль. Студенты после выполнения соответствующих (первому или второму модулю) практических и лабораторных работ допускаются к рубежному контролю. Каждый из двух рубежных контролей (модулей) оценивается по **30** балльной шкале.

Итоговый контроль. Итоговый контроль реализуется в форме защиты собственно созданных программ (в виде компьютерного тестирования) и оценивается по **30** балльной шкале.

Правила оценивания рубежного и итогового контроля.

8 Критерий оценки знаний студентов на экзамене

Выставление оценок на экзаменах осуществляется на основе принципов объективности, справедливости, всестороннего анализа качества знаний студентов, и других положений, способствующих повышению надежности оценки знаний обучающихся, и устранению субъективных факторов.

В соответствии с действующими нормативными актами и рекомендациями Министерства образования и науки КР устанавливаются следующие критерии выставления оценок на экзаменах:

- ***оценка "отлично"*** выставляется студенту, который обнаружил на экзамене всестороннее, систематическое и глубокое знание учебно-программного матери-ала, умение свободно выполнять задания, предусмотренные программой, который усвоил основную литературу и ознакомился с дополнительной литературой, рекомендованной программой. Как правило, оценка "отлично" выставляется студентам, усвоившим взаимосвязь основных понятий дисциплины и их значений для приобретаемой профессии, проявившим творческие способности в понимании, изложении и использовании учебно-программного материала;

- ***оценка "хорошо"*** выставляется студенту, который на экзамене обнаружил полное знание учебно-программного материала, успешно выполнил предусмотренные в программе задания, усвоил основную литературу, рекомендованную в программе. Как правило, оценка

"хорошо" выставляется студентам, показавшим систематический характер знаний по дисциплине и способным к их самостоятельному выполнению и обновлению в ходе дальнейшей учебной работы и профессиональной деятельности;

- ***оценка "удовлетворительно"*** выставляется студенту, обнаружившему знание основного учебного материала в объеме, необходимом для дальнейшей учебы и предстоящей работы по профессии, справляющемуся с выполнением заданий, предусмотренных программой, который ознакомился с основной литературой, рекомендованной программой. Как правило, оценка "удовлетворительно" выставляется студентам, допустившим погрешности в ответе на экзамене и при выполнении экзаменационных заданий, но обладающим необходимыми знаниями для их устранения под руководством преподавателя;

- ***оценка "неудовлетворительно"*** выставляется студенту, обнаружившему пробелы в знаниях основного учебно-программного материала, допустившему принципиальные ошибки в выполнении предусмотренных программой заданий, не ознакомившемуся с основной литературой, предусмотренной программой, и не овладевшему базовыми знаниями, предусмотренными по данной дисциплине и определенными соответствующей программой курса (перечень основных знаний и умений, которыми должны овладеть студенты, является обязательным элементом рабочей программы курса).

8.1. Оценка знаний (академической успеваемости) осуществляется по 100 балльной системе (шкале) следующим образом:

30 балльная система	100 балльная система	Оценка по буквенной системе	Цифровой эквивалент оценки по GPA	Оценка по традиционной системе
26 - 30	87 – 100	A	4,0	Отлично
24 - 25	80 – 86	B	3,33	Хорошо
22 - 23	74 – 79	C	3,0	
20 - 21	68 - 73	Д	2,33	Удовлетворительно
18 - 19	60 – 67	E	2,0	
9 - 17	31 - 60	FX	0	Неудовлетворительно
0 - 8	0 - 30	F	0	

8.2. Технологическая карта дисциплины

Всего	Ауд. часы	СРС, СРСП	1-модуль (90 ч., 30 б.)				2-модуль (90 ч., 30 б.)				Итоговый контроль (ЖТ) (30 б.)				П	
			Ауд. ч.		СРС, СРСП	Рубежный контроль 1	Ауд. ч.		СРС, СРСП	Рубежный	Лекция	Лабораторн	СРС	**Итоговый контроль**		
			Лекция	Практические занятия			Лекция	Практические занятия								
90	45	45	15	30	45		15	30	45							
Баллы				**30**	**30**	**30 б.**		**45**	**30**	**30 б.**	**30**	**30**	**30**	**30 б**	**10 б**	
Итоги модулей			ТК=(Лек+Лаб+ +СРС)/3, М1=(ТК1+ТК2+ +РК1)/3				ТК=(Лек+Лаб+ +СРС)/3, М2=(ТК3+ТК4+ +РК2)/3				ИК=(Лек+Лаб+ +СРС)/3, Экз=М1+М2+ИК+П					**100**

9. Тематический план и содержание учебной дисциплины «Программная инженерия » для студентов очного и заочного обучения

№	Темы лекционных занятий	О/О	З/О
1.	**Введение в программную инженерию:** цели, задачи, история, стандарты.	2	2
2.	**Жизненный цикл программного обеспечения (SDLC):** основные этапы и модели.	2	СРС
3.	**Методологии разработки ПО:** каскадная, спиральная, V-модель, Agile, Scrum.	2	2
4.	**Инженерия требований:** сбор, анализ, спецификация и управление изменениями.	2	2

5.	**Моделирование программных систем:** UML, диаграммы вариантов использования, классов, активности.	2	СРС
6.	**Проектирование архитектуры программного обеспечения:** слоистая архитектура, MVC, микросервисы.	2	СРС
7.	**Управление программными проектами:** планирование, ресурсы, сроки, риски.	2	СРС
8.	**Контроль качества и тестирование ПО:** виды тестирования, методики, тест-дизайн.	2	СРС
9.	**Средства и инструменты программной инженерии:** IDE, Git, CI/CD, Jira, Docker.	**2**	**2**
10.	**Управление версиями и конфигурацией ПО:** Git, DevOps, репозиторные практики.	2	2
11.	**Документирование программных решений:** техническое задание, пользовательская и проектная документация.	**2**	СРС
12.	**Обеспечение надёжности и отказоустойчивости ПО:** принципы, методы, анализ ошибок.	2	СРС
13.	**Информационная безопасность в программной инженерии:** уязвимости, шифрование, защита данных.	**2**	СРС
14.	**Анализ и оценка качества программного обеспечения:** метрики, модели, стандарты (ISO/IEC 25010).	2	СРС
15.	**Современные тенденции программной инженерии:** DevOps, No-code/Low-code, AI-инжиниринг, цифровые двойники.	**2**	СРС
		30	**10**

№	Темы практических занятий	О/О	З/О
1.	**Анализ программных профессий и отраслей применения ПО.** *Задание:* составить карту профессий в сфере разработки и выбрать 2 реальные ИТ-проекта для анализа.	2	2
2.	**Построение диаграммы жизненного цикла ПО (SDLC).** *Задание:* разобрать и визуализировать SDLC по конкретному примеру (например, разработка сайта школы).	2	СРС
3.	**Сравнение методологий разработки: Waterfall vs Agile.** *Задание:* создать таблицу сравнений и обсудить преимущества в разных ситуациях.	2	2
4.	**Сбор требований от "заказчика".** *Задание:* в парах: один — "заказчик", другой — "разработчик". Составить список пользовательских требований.	2	2
5.	**Составление спецификации требований (SRS).** *Задание:* оформить требования в виде шаблона документа (стандарт IEEE/ГОСТ).	2	СРС
6.	**Построение диаграммы вариантов использования (Use Case).** *Задание:* смоделировать функциональность учебного портала.	2	СРС
7.	**Создание UML-диаграмм классов.** *Задание:* описать предметную область (например, библиотека, интернет-магазин) через классы и связи.	2	СРС
8.	**Разработка архитектурной схемы ПО.** *Задание:* спроектировать	2	СРС

	трёхуровневую архитектуру (клиент–сервер–БД) с описанием слоёв.		
9.	**Планирование программного проекта.** *Задание:* создать диаграмму Ганта, определить этапы, ресурсы, сроки.	**2**	**2**
10.	**Идентификация рисков в проекте.** *Задание:* провести SWOT-анализ для ИТ-проекта и предложить план управления рисками.	2	2
11.	**Разработка тест-кейсов и тест-планов.** *Задание:* составить примеры функциональных и нефункциональных тестов.	**2**	СРС
12.	**Практика модульного тестирования (unit-тесты).** *Задание:* написать тесты на Python или Java для простых функций.	2	СРС
13.	**Работа с Git: инициализация, коммиты, ветвление.** *Задание:* выполнить действия с репозиторием на GitHub.	**2**	СРС
14.	**Настройка CI/CD пайплайна (теория + демонстрация в GitHub Actions или GitLab CI).** *Задание:* автоматизация сборки и тестов простого проекта.	2	СРС
15.	**Использование Jira или Trello для ведения проекта.** *Задание:* создать доску задач, разметить backlog, спринт.	**2**	СРС
16.	**Разработка шаблона технического задания (ТЗ).** *Задание:* составить ТЗ для учебного проекта (например, чат-приложения).	**2**	2

17.	**Обзор и применение стандартов качества (ISO/IEC 25010).** *Задание:* провести самооценку качества ПО по 3 критериям (например, надёжность, удобство, производительность).	**2**	2
18.	**Поиск уязвимостей и составление рекомендаций по защите.** *Задание:* рассмотреть кейс (например, небезопасная форма входа) и предложить защиту.	**4**	2
19.	**Разработка микропроекта по методологии Agile (эмуляция 1 спринта).** *Задание:* провести планирование, разделить задачи, презентовать результат.	**3**	2
20.	**Презентация проекта с применением современных тенденций (DevOps, AI-интеграция).** *Задание:* защита своего проекта с акцентом на используемые современные подходы.	**4**	2
		45	**20**

Темы для СРС (Самостоятельная работа студента):

1. Анализ эволюции моделей жизненного цикла ПО
2. Сравнительная характеристика Agile, Scrum, Kanban и Waterfall
3. Составление обзора международных стандартов в программной инженерии (ISO, IEEE, ГОСТ)
4. Разработка пользовательских историй (User Stories) для заданной задачи
5. Сравнительный анализ инструментов UML-моделирования (Draw.io, StarUML, Visual Paradigm)
6. Особенности архитектуры микросервисов и их преимущества
7. Методы управления проектами в ИТ (PMBOK, PRINCE2)
8. Современные подходы к обеспечению надёжности программ

9. Инструменты статического и динамического анализа кода
10. Использование GitHub в командной разработке
11. Обзор платформ для CI/CD: Jenkins, GitLab CI, GitHub Actions
12. Документирование программного продукта: структура и содержание
13. Методы и средства автоматизации тестирования
14. Анализ реального инцидента ИБ (утечка, взлом) и рекомендации по защите
15. Тенденции развития программной инженерии: DevOps, AI-сопровождение, Low-code

Темы для СРСП (под руководством преподавателя):

1. Подготовка технического задания на создание информационной системы
2. Разработка спецификации требований (SRS) для учебного проекта
3. Моделирование проекта с использованием UML (варианты использования, диаграммы классов)
4. Разработка архитектурной схемы будущего программного продукта
5. Разработка прототипа интерфейса пользователя с пояснениями
6. Создание и оформление диаграммы Ганта с детализацией задач
7. Подготовка и реализация тест-плана для конкретного модуля программы
8. Проведение модульного тестирования и составление отчёта о покрытии
9. Настройка репозитория проекта с ветвлением, pull-request и ревью
10. Создание пайплайна CI/CD для автоматической сборки и тестирования
11. Подготовка пользовательской документации (инструкция, справка)
12. Разработка и защита мини-проекта по созданию веб-приложения с полным циклом
13. Анализ рисков и подготовка плана их минимизации для проекта
14. Проведение Code Review с аннотацией ошибок и предложениями улучшений
15. Финальная презентация проекта с демонстрацией архитектуры, кода и документации

10. Учебно-методическое, информационное и материально-техническое обеспечение дисциплины

Учебные пособия и методические материалы

Основные учебные пособия:

1. **В.В. Липаев. "Программная инженерия сложных заказных программных продуктов"**
   Учебное пособие, охватывающее методы и процессы проектирования и производства сложных заказных программных продуктов для технических систем реального времени.
   Скачать PDF
2. **А.И. Долженко, С.А. Глушенко. "Программная инженерия"**
   Учебное пособие, посвященное вопросам теории и инструментальной поддержке программной инженерии, включая модели жизненного цикла и методологии создания программного обеспечения.
   Скачать PDF
3. **Л.С. Носова. "Основы программной инженерии"**
   Учебно-методическое пособие, нацеленное на формирование у обучающихся профессиональных компетенций в области программной инженерии.
   Скачать PDF
4. **Олеслав Антамошкин. "Программная инженерия. Теория и практика"**
   Учебник, освещающий современные методы и средства программной инженерии, детально рассматривающий процесс разработки программного обеспечения.
   Скачать PDF

Методические материалы:

6. **С.А. Орлов. "Основы программной инженерии"**
   Учебник, содержащий теоретические материалы и практические задания по программной инженерии.
   Скачать PDF

11. Вопросы для подготовки к экзамену

Экзаменационные вопросы

1. **Что такое Жизненный цикл ПО?**
   Жизненный цикл ПО включает стадии: анализ требований, проектирование, реализация, тестирование, сопровождение.
2. **Что такое Agile?**
   Agile — гибкая методология разработки с итеративным подходом и постоянной обратной связью с заказчиком.
3. **Что такое Scrum?**
   Scrum — фреймворк Agile с

ролями Product Owner, Scrum Master и спринтами.

4. **Что такое Waterfall?**
Waterfall — каскадная модель разработки ПО, где этапы выполняются последовательно.

5. **Что такое Сбор требований?**
На этом этапе выявляются нужды заказчика и формируются технические требования.

6. **Что такое UML?**
UML — универсальный язык моделирования, используется для визуального описания систем.

7. **Что такое Use Case диаграмма?**
Диаграмма вариантов использования описывает взаимодействие пользователей с системой.

8. **Что такое Диаграмма классов?**
Диаграмма классов показывает структуру объектов, их свойства и связи.

9. **Что такое Тестирование ПО?**
Тестирование — процесс проверки соответствия программы требованиям и выявления ошибок.

10. **Что такое Unit-тестирование?**
Модульное тестирование отдельных функций или компонентов программы.

11. **Что такое Интеграционное тестирование?**
Проверка взаимодействия между компонентами системы.

12. **Что такое Функциональное тестирование?**
Проверка, как ПО выполняет заявленные функции.

13. **Что такое Git?**
Git — распределённая система контроля версий, позволяет отслеживать изменения в коде.

14. **Что такое CI/CD?**
CI/CD — непрерывная интеграция и доставка: автоматизация сборки, тестирования и развертывания ПО.

15. **Что такое DevOps?**
DevOps — культура и практика объединения разработки и эксплуатации для быстрой поставки ПО.

16. **Что такое Документирование в программной инженерии?**
Документация описывает требования, архитектуру, код, инструкции пользователя.

17. **Что такое UAT?**
UAT (приёмочное тестирование) — проверка системы конечными пользователями.

18. **Что такое Архитектура ПО?**
Архитектура описывает общую структуру и взаимодействие компонентов системы.

19. **Что такое MVC?**
MVC — архитектурный шаблон с разделением: Модель, Представление и Контроллер.

20. **Что такое Обеспечение качества ПО?**
Процесс обеспечения соответствия программного продукта стандартам и ожиданиям.
21. **Что такое ролевая модель в Scrum?**
В Scrum выделяются роли: Scrum Master, Product Owner и команда разработчиков.
22. **Что такое Product Backlog?**
Список всех требований, функций и задач, которые нужно реализовать в продукте.
23. **Что такое Sprint?**
Короткий цикл разработки (обычно 1–4 недели), в конце которого создаётся работающий продукт.
24. **Что такое Daily Scrum?**
Короткая ежедневная встреча команды для координации и планирования.
25. **Что такое Refactoring?**
Изменение внутренней структуры кода без изменения внешнего поведения программы.
26. **Что такое Pull Request?**
Запрос на внесение изменений в репозиторий, обычно используется для ревью кода.
27. **Что такое Merge Conflict?**
Конфликт, возникающий при объединении изменений из разных веток в Git.
28. **Что такое CI Pipeline?**
Набор автоматических шагов: сборка, тестирование, проверка кода и деплой.
29. **Что такое Issue Tracker?**
Инструмент для учёта задач, багов и пожеланий (например, Jira, Trello, YouTrack).
30. **Что такое Version Control System (VCS)?**
Система управления версиями — программный инструмент для отслеживания изменений в коде.
31. **Что такое Feature Branch?**
Отдельная ветка в Git, предназначенная для разработки новой функциональности.
32. **Что такое Code Review?**
Процесс проверки кода другими разработчиками перед объединением изменений.
33. **Что такое Dependency Injection?**
Паттерн, при котором зависимости внедряются извне, а не создаются внутри объекта.
34. **Что такое REST API?**
Интерфейс взаимодействия по протоколу HTTP с использованием CRUD-операций и стандартных URL.
35. **Что такое JSON?**
Формат обмена данными между клиентом и сервером, основанный на JavaScript.
36. **Что такое Regression Testing?**
Повторное тестирование после

изменений, чтобы убедиться, что старая функциональность не сломана.

37. **Что такое Smoke Testing?**
Быстрая проверка основных функций ПО для оценки его работоспособности.

38. **Что такое Sprint Retrospective?**
Встреча команды Scrum по итогам спринта для анализа и улучшения процесса.

39. **Что такое Sprint Planning?**
Планирование задач и целей спринта до его начала.

40. **Что такое Technical Debt?**
Проблемы, отложенные при разработке в пользу скорости, требующие исправления позже.

41. **Что такое Continuous Deployment?**
Автоматическое развертывание кода в продакшн после прохождения всех проверок.

42. **Что такое Kanban?**
Метод управления задачами на основе визуального потока и ограничения количества задач в работе.

43. **Что такое User Story?**
Краткое описание функционала с точки зрения пользователя: «Как [роль] я хочу [цель], чтобы [результат]».

44. **Что такое Acceptance Criteria?**
Условия, при которых пользовательская история считается завершённой и принятой.

45. **Что такое Mock-объект?**
Подставной объект, имитирующий поведение настоящего компонента для тестирования.

46. **Что такое Test Coverage?**
Показатель, насколько полно тесты покрывают код (по строкам, ветвлениям и т.д.).

47. **Что такое Static Code Analysis?**
Анализ кода без его запуска для выявления ошибок и нарушений стандартов.

48. **Что такое Software Requirement Specification (SRS)?**
Документ, содержащий все функциональные и нефункциональные требования к программному продукту.

49. **Что такое Архитектурный шаблон?**
Готовая структура проектирования, например MVC, Layered, Microservices.

50. **Что такое Use Case?**
Сценарий взаимодействия пользователя с системой для достижения цели.

51. **Что такое Load Testing?**
Тестирование производительности при ожидаемой нагрузке.

52. **Что такое Stress Testing?**
Проверка поведения системы при экстремальных условиях (нагрузка, отказ компонентов и т.д.).
53. **Что такое Release Management?**
Процесс подготовки, тестирования и публикации версий программного продукта.
54. **Что такое Баг-репорт?**
Официальное сообщение об ошибке, включающее шаги воспроизведения и ожидаемое поведение.
55. **Что такое CI-сервер?**
Инструмент, автоматически запускающий сборку и тесты при каждом обновлении кода (например, Jenkins).
56. **Что такое Branching Strategy?**
Правила создания, именования и слияния веток в системе контроля версий.
57. **Что такое Архитектура «Клиент-Сервер»?**
Модель взаимодействия, в которой клиент запрашивает услуги у сервера.
58. **Что такое Software Design Pattern?**
Повторяемое решение архитектурной или кодовой задачи (например, Singleton, Observer).
59. **Что такое Code Smell?**
Признак возможной проблемы в коде: избыточность, дублирование, запутанность.
60. **Что такое Build Automation?**
Автоматизация процесса сборки программы: компиляция, упаковка, запуск тестов.
61. **Что такое Ветка (branch) в Git?**
Отдельная линия разработки, в которой можно работать над новыми функциями без влияния на основную ветку.
62. **Что такое Merge в Git?**
Операция объединения изменений из одной ветки в другую.
63. **Что такое Rebase в Git?**
Перепроигрывание изменений поверх другой ветки, для создания «чистой» истории.
64. **Что такое Bugfix?**
Исправление ошибки в программном обеспечении.
65. **Что такое Feature Freeze?**
Этап, когда запрещается добавлять новые функции, разрешены только исправления багов.
66. **Что такое CI Tool?**
Инструмент непрерывной интеграции (например, GitLab CI, Jenkins, Travis CI).
67. **Что такое IDE?**
Интегрированная среда разработки, объединяющая

редактор, компилятор и отладчик (например, VS Code, IntelliJ).

68. **Что такое Static Typing?**
Типы данных переменных определяются до выполнения программы (например, в Java, C++).

69. **Что такое Dynamic Typing?**
Типы переменных определяются во время выполнения (например, Python, JavaScript).

70. **Что такое Микросервисная архитектура?**
Разделение приложения на независимые сервисы, каждый из которых выполняет одну задачу.

71. **Что такое Monolith?**
Цельное приложение, в котором все функции объединены в одном исполняемом блоке.

72. **Что такое База данных?**
Структурированное хранилище информации, с которой работает программа.

73. **Что такое API?**
Интерфейс программирования приложений — набор правил для взаимодействия между программами.

74. **Что такое REST?**
Стиль архитектуры API, использующий стандартные HTTP-запросы (GET, POST и др.).

75. **Что такое CRUD?**
Операции создания, чтения, обновления и удаления данных.

76. **Что такое Инкапсуляция?**
Сокрытие внутренней реализации объекта от внешнего мира.

77. **Что такое Наследование в ООП?**
Механизм, при котором один класс перенимает свойства и методы другого.

78. **Что такое Полиморфизм?**
Способность объектов вести себя по-разному в зависимости от контекста.

79. **Что такое Класс?**
Шаблон (структура) для создания объектов с определёнными свойствами и методами.

80. **Что такое Объект?**
Конкретный экземпляр класса с определённым состоянием.

81. **Что такое Agile-манифест?**
Набор ценностей и принципов гибкой разработки программного обеспечения.

82. **Что такое TDD?**
Разработка через тестирование: сначала пишется тест, потом код.

83. **Что такое BDD?**
Разработка через поведение: спецификация требований в виде понятных сценариев.

84. **Что такое Mocking?**
Создание заглушек для зависимостей во время тестирования.

85. **Что такое Рефакторинг?**
Изменение внутренней структуры кода без изменения поведения.

86. **Что такое User Flow?**
Путь, по которому пользователь проходит в приложении для достижения цели.
87. **Что такое MVP (Minimum Viable Product)?**
Минимально жизнеспособный продукт с основным функционалом для тестирования идеи.
88. **Что такое UX?**
Пользовательский опыт — насколько удобно и эффективно пользователь взаимодействует с программой.
89. **Что такое UI?**
Пользовательский интерфейс — визуальная часть взаимодействия с программой.
90. **Что такое Git Commit?**
Фиксация изменений в истории проекта.
91. **Что такое Git Push?**
Отправка локальных изменений в удалённый репозиторий.
92. **Что такое Git Pull?**
Получение изменений из удалённого репозитория.
93. **Что такое Rollback?**
Откат системы к предыдущей стабильной версии.
94. **Что такое Continuous Integration?**
Автоматическая проверка и сборка кода при каждом изменении.
95. **Что такое Continuous Delivery?**
Готовность кода к релизу в любой момент после успешной сборки.
96. **Что такое Artifact?**
Готовый результат сборки (например, .jar-файл, .exe, архив).
97. **Что такое Codebase?**
Совокупность исходного кода проекта.
98. **Что такое Coding Standards?**
Соглашения по стилю и структуре написания кода.
99. **Что такое Pair Programming?**
Практика совместной разработки двумя программистами за одним компьютером.
100. **Что такое Release Notes?**
Официальный список изменений, включённых в новую версию продукта.
101. **Что такое Story Point?**
Единица измерения относительной сложности задачи в Agile.
102. **Что такое Burndown Chart?**
График, показывающий объём оставшейся работы в спринте.
103. **Что такое Sprint Goal?**
Основная цель, которую команда должна достичь в текущем спринте.
104. **Что такое Velocity в Scrum?**
Количество story points, которое команда успевает выполнять за один спринт.

105. **Что такое Epic в Agile?**
Крупная пользовательская история, состоящая из нескольких user stories.
106. **Что такое Definition of Done (DoD)?**
Критерии, при которых задача считается завершённой.
107. **Что такое Acceptance Testing?**
Тестирование для проверки соответствия продукта ожиданиям клиента.
108. **Что такое Sandbox Environment?**
Изолированная среда для безопасного тестирования и экспериментов.
109. **Что такое Software Maintenance?**
Процесс исправления, улучшения и адаптации ПО после выпуска.
110. **Что такое Патч (Patch)?**
Обновление программного обеспечения, исправляющее ошибки или уязвимости.
111. **Что такое Refactoring Tool?**
Инструмент, помогающий улучшить структуру кода без изменения его поведения.
112. **Что такое Bug Tracking System?**
Система для регистрации, отслеживания и управления ошибками (например, Bugzilla, Jira).
113. **Что такое Feature Toggle?**
Техника включения/отключения функциональности без изменения кода.
114. **Что такое Sprint Backlog?**
Список задач, запланированных для выполнения в текущем спринте.
115. **Что такое Technical Specification?**
Документ, описывающий внутренние аспекты реализации ПО.
116. **Что такое Unit?**
Минимальная тестируемая часть программы (обычно функция или метод).
117. **Что такое API Documentation?**
Документация, объясняющая, как использовать API.
118. **Что такое Software Modeling?**
Процесс создания абстрактных моделей программной системы.
119. **Что такое Design Thinking?**
Подход к решению проблем с акцентом на потребности пользователя.
120. **Что такое Software Reuse?**
Повторное использование существующего кода или компонентов в новых проектах.
121. **Что такое Use Case Scenario?**
Подробный пошаговый пример

использования системы для достижения цели.

122. **Что такое Software Audit?**
Проверка соответствия ПО стандартам, требованиям и безопасности.

123. **Что такое Software Metrics?**
Численные показатели, оценивающие качество и производительность кода.

124. **Что такое Code Refactoring?**
Перестройка существующего кода для повышения его читаемости и сопровождения.

125. **Что такое Sprint Review?**
Встреча после спринта для демонстрации результата и обсуждения улучшений.

126. **Что такое Continuous Monitoring?**
Постоянный сбор и анализ данных о работе приложения в реальном времени.

127. **Что такое Software Testing Life Cycle (STLC)?**
Процесс, включающий планирование, разработку тестов, выполнение и завершение.

128. **Что такое Branch Protection Rule?**
Ограничения в системе контроля версий для защиты важных веток.

129. **Что такое Merge Request?**
Запрос на объединение изменений из одной ветки в другую, с возможностью ревью.

130. **Что такое Software Development Kit (SDK)?**
Набор инструментов и библиотек для создания приложений под определённую платформу.

131. **Что такое Open Source?**
Программное обеспечение с открытым исходным кодом, доступным для модификации.

132. **Что такое Proprietary Software?**
Закрытое ПО, которое нельзя изменять и распространять без разрешения.

133. **Что такое Test Case?**
Документ, описывающий входные данные, действия и ожидаемый результат.

134. **Что такое Smoke Test?**
Минимальный набор тестов, проверяющих базовую работоспособность системы.

135. **Что такое Issue в GitHub?**
Элемент отслеживания задач, ошибок или предложений в проекте.

136. **Что такое Hotfix?**
Срочное исправление критической ошибки в продакшн-версии.

137. **Что такое Software Architecture Document?**
Описывает архитектурные решения, структуры и шаблоны проекта.

138. **Что такое Change Request?**
Официальное предложение внести изменения в проект.
139. **Что такое Software Reliability?**
Способность ПО выполнять заданные функции при определённых условиях.
140. **Что такое Build Server?**
Сервер, автоматически выполняющий сборку и тестирование кода проекта.
141. **Что такое SLA (Service Level Agreement)?**
Документ, определяющий уровень качества и доступности программного сервиса.
142. **Что такое Software Scalability?**
Способность ПО сохранять производительность при росте нагрузки.
143. **Что такое Logging?**
Запись событий и сообщений во время работы программы для последующего анализа.
144. **Что такое Monitoring?**
Отслеживание работы системы в реальном времени для предупреждения сбоев.
145. **Что такое Software Usability?**
Удобство использования программы для конечного пользователя.
146. **Что такое Rollout Strategy?**
План поэтапного внедрения новой версии ПО.
147. **Что такое Fallback?**
Механизм отката к безопасной или предыдущей версии при ошибке.
148. **Что такое A/B Testing?**
Сравнение двух версий интерфейса или функционала для оценки лучшего варианта.
149. **Что такое Stakeholder?**
Любое лицо или организация, заинтересованное в результате проекта.
150. **Что такое Build Artifact?**
Файл, получаемый в результате сборки проекта (например, .jar, .exe).
151. **Что такое Binary Repository?**
Хранилище готовых сборок программ (например, Artifactory, Nexus).
152. **Что такое Software Localization?**
Адаптация интерфейса и логики ПО под конкретный язык и культуру.
153. **Что такое Scrum Board?**
Визуальный инструмент для отслеживания задач спринта.
154. **Что такое Software Compliance?**
Соответствие программного обеспечения нормативным и правовым требованиям.

155. **Что такое Incident Management?**
Процесс выявления, регистрации и устранения инцидентов в работе ПО.
156. **Что такое Root Cause Analysis?**
Анализ первопричины проблемы для предотвращения повторения.
157. **Что такое Software Lifecycle Management (SLM)?**
Комплексное управление всеми этапами жизни ПО — от идеи до удаления.
158. **Что такое Release Candidate (RC)?**
Версия, готовая к выпуску, при отсутствии критических ошибок.
159. **Что такое Exploratory Testing?**
Интерактивное, творческое тестирование без строгого плана.
160. **Что такое Pair Testing?**
Два специалиста совместно проводят тестирование ПО.
161. **Что такое Feature Branch Workflow?**
Модель разработки, где каждая новая функция создаётся в отдельной ветке.
162. **Что такое Secure Coding?**
Писать код с учётом защиты от типичных уязвимостей (SQL-инъекции, XSS и др.).
163. **Что такое Vulnerability Scanner?**
Инструмент, проверяющий ПО на наличие известных уязвимостей.
164. **Что такое QA (Quality Assurance)?**
Комплекс мероприятий по обеспечению высокого качества разработки.
165. **Что такое Test Plan?**
Документ, определяющий объём, методы и ресурсы для тестирования.
166. **Что такое Load Balancing?**
Распределение нагрузки между серверами для повышения надёжности и скорости.
167. **Что такое Postmortem Report?**
Аналитический отчёт о причинах сбоя после его устранения.
168. **Что такое Code Linting?**
Автоматическая проверка кода на соответствие стилю и потенциальные ошибки.
169. **Что такое Code Formatting?**
Приведение кода к стандартному виду (отступы, скобки, пробелы и т.д.).
170. **Что такое Software Deployment?**
Процесс установки и запуска программного обеспечения в целевой среде.
171. **Что такое Blue-Green Deployment?**
Метод обновления системы путём переключения между двумя идентичными средами.

172. **Что такое Canary Release?**
Постепенный выпуск новой версии для ограниченной аудитории.
173. **Что такое Rollback Plan?**
План отката к предыдущей версии в случае неудачного релиза.
174. **Что такое Dev Environment?**
Среда разработки, где пишется и тестируется код до его релиза.
175. **Что такое Staging Environment?**
Тестовая среда, максимально приближенная к продакшн.
176. **Что такое Production Environment?**
Рабочая среда, в которой находится конечный продукт для пользователей.
177. **Что такое Functional Specification?**
Документ, описывающий функциональные требования к системе.
178. **Что такое Non-functional Requirements?**
Требования к надёжности, производительности, безопасности, удобству и др.
179. **Что такое ISO/IEC 25010?**
Международный стандарт, описывающий характеристики качества ПО.
180. **Что такое Continuous Improvement?**
Постоянное совершенствование процессов и качества продукта.
181. **Что такое Zero Downtime Deployment?**
Обновление без остановки сервиса для пользователей.
182. **Что такое Dead Code?**
Код, который не выполняется и не влияет на работу программы.
183. **Что такое Code Ownership?**
Ответственность конкретного разработчика или команды за участок кода.
184. **Что такое Software Portability?**
Способность ПО работать на разных платформах без модификаций.
185. **Что такое Maintainability?**
Лёгкость поддержки, доработки и устранения ошибок в программе.
186. **Что такое Test Automation?**
Автоматизация запуска тестов для ускорения проверки качества.
187. **Что такое Cross-browser Testing?**
Проверка корректности работы веб-приложения в разных браузерах.
188. **Что такое Regression Suite?**
Набор тестов для проверки, что новая версия не нарушила старую функциональность.
189. **Что такое Software Engineering?**
Инженерный подход к разработке,

тестированию и сопровождению ПО.

190. **Что такое Software Development Lifecycle (SDLC)?**
Процесс, охватывающий все стадии от идеи до завершения использования ПО.

191. **Что такое Agile Estimation?**
Оценка задач с помощью story points, planning poker и других инструментов.

192. **Что такое Test-Driven Development (TDD)?**
Методология, при которой сначала пишутся тесты, затем — код.

193. **Что такое QA Engineer?**
Специалист по обеспечению качества: тестирование, документация, автоматизация.

194. **Что такое Unit Test Framework?**
Библиотека или инструмент для написания и запуска модульных тестов (например, JUnit, PyTest).

195. **Что такое Issue Tracking System?**
Система учёта и отслеживания задач и ошибок (например, Jira, YouTrack).

196. **Что такое Agile Coach?**
Специалист, обучающий команду Agile-практикам и помогающий внедрять их.

197. **Что такое Bug Lifecycle?**
Этапы жизни ошибки: обнаружение, регистрация, подтверждение, исправление, закрытие.

198. **Что такое Software Testing Pyramid?**
Модель, рекомендующая больше модульных тестов, меньше — интеграционных и UI.

199. **Что такое Code Smell?**
Признаки плохой архитектуры или стиля кода, которые могут привести к ошибкам.

200. **Что такое Feature Request?**
Запрос пользователя на добавление нового функционала.

12. Методические указания для преподавателей дисциплины «Программная инженерия»

Общие положения

Курс «Программная инженерия» является базовой профессиональной дисциплиной для студентов, обучающихся по направлениям подготовки в области информационных технологий. Цель преподавания — сформировать у студентов системное понимание жизненного цикла программного обеспечения, научить применять современные методы и технологии разработки, тестирования и сопровождения программных продуктов.

2. Цели преподавания

- Формирование представления о программной инженерии как об инженерной дисциплине.
- Ознакомление с моделями жизненного цикла ПО, методологиями разработки (Waterfall, Agile, Scrum и др.).
- Развитие навыков анализа требований, проектирования, кодирования и тестирования.
- Освоение инструментов управления версиями, автоматизации разработки (CI/CD), сопровождения и документации ПО.

3. Задачи преподавателя

- Организовать последовательное изучение теоретических и практических аспектов дисциплины.
- Формировать у студентов умения работать в команде, вести проекты и применять современные технологии.
- Развивать способности к анализу, критическому мышлению и принятию инженерных решений.
- Оценивать прогресс студентов через отчётность, проектные задания, тестирование и защите мини-проектов.

4. Формы организации занятий

- **Лекции:** изложение теоретических аспектов, демонстрация примеров, использование мультимедиа.
- **Практические занятия:** решение задач, анализ кейсов, работа с диаграммами и тест-кейсами.
- **Лабораторные работы:** работа в IDE, Git, моделирование в UML, написание и тестирование кода.
- **СРС / СРСП:** выполнение мини-проектов, подготовка технической документации, самообучение по заданной теме.

5. Методы обучения

- Объяснительно-иллюстративный
- Проблемный и проектный
- Практико-ориентированный (работа в GitHub, Jira, CI/CD-средах)
- Использование кейс-стади и симуляции Agile-команд

6. Контроль знаний студентов

Контроль осуществляется по следующим видам:

- текущие опросы и мини-тесты на лекциях;

- оценка выполнения лабораторных и практических заданий;
- защита проектов, обсуждение проектных решений;
- итоговый экзамен / зачет (по билетам, тестам или защите проекта).

7. Методические рекомендации

- Акцентировать внимание на инженерной составляющей (анализ, проектирование, сопровождение ПО).
- Давать студентам возможность реализовать мини-проекты (например, чат, ToDo-приложение, REST-сервис).
- Использовать реальные инструменты: Git, Trello/Jira, Visual Studio Code, Docker, GitHub Actions.
- Обсуждать практические кейсы из ИТ-индустрии (внедрение Agile, инциденты, фичи и баги).
- Внедрять элементы командной работы и презентации результатов на зачёте.

8. Рекомендуемые ресурсы

- Липаев В.В. *Программная инженерия сложных программных систем*
- Sommerville I. *Software Engineering*
- Гусева А.И. *Программная инженерия: учебное пособие*
- GitHub Education, Atlassian Agile Coach
- Онлайн-курсы Coursera, Stepik, GitLab Learn по DevOps/CI

График проведенных занятий

№	Дата проведения практических занятий по расписанию	Дата проведения лабораторных занятий по расписанию	Дата проведения СРСП по расписанию
1.			
2.			
3.			
4.			
5.			
6.			
7.			
8.			
9.			
10.			
11.			
12.			
13.			
14.			
15.			
16.			
17.			
18.			
19.			
20.			
21.			
22.			
23.			
24.			
25.			
26.			
27.			
28.			
29.			
30.			