Методические указания

по выполнению практических работ по дисциплине «Современные технологии разработки программного обеспечения»

Электронное учебное издание

Методические указания по выполнению лабораторных работ по дисциплине «Современные технологии разработки программного обеспечения». Электронное учебное издание.

Учебное издание содержит указания и требования к порядку выполнения лабораторных работ. Дается теоретический материал, необходимый для выполнения работ, и предлагается пример реализации программ, аналогичных разрабатываемым на лабораторных работах. Определяются цели и объем, требования к отчетам, а также приводятся варианты заданий и примерный список контрольных вопросов для защиты выполненных заданий.

Для студентов 1 курса магистратуры специальности «Прикладная информатика».

Электронное учебное издание

Оглавление

ВВЕДЕНИЕ **6**

1 ОБЩЕЕ ОПИСАНИЕ ПРАКТИКУМА **7**

1.1 Взаимосвязь элементов практикума 8

1.2 Общие требования к отчету 8

1.3 Требование к стилю кодирования 9

1.4 Установка и настройка GNU/Linux Ubuntu 18.04 LTS 9

1.5 Использование репозитория исходного кода 11

2 ЛАБОРАТОРНАЯ РАБОТА №1. РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ В МНОГОУРОВНЕВОЙ АРХИТЕКТУРЕ С ИСПОЛЬЗОВАНИЕМ УДАЛЁННОГО РЕПОЗИТОРИЯ КОДА **12**

2.1 Изоляция предметной области 12

2.2 Порядок выполнения работы 14

2.3 Варианты заданий 21

2.4 Требования к отчету 22

2.5 Контрольные вопросы 22

3 ЛАБОРАТОРНАЯ РАБОТА №2. РЕАЛИЗАЦИЯ МОДЕЛИ ПРЕДМЕТНОЙ ОБЛАСТИ С ИСПОЛЬЗОВАНИЕМ ТЕХНИКИ НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ **23**

3.1 MDD, структура модели, углубляющий рефакторинг 23

3.2 Порядок выполнения работы 25

3.3 Варианты заданий 26

3.4 Требования к отчету 29

3.5 Контрольные вопросы 30

4 ЛАБОРАТОРНАЯ РАБОТА №3. РАЗРАБОТКА ПРОСТОГО DEVOPS-ПРИЛОЖЕНИЯ **31**

4.1 Введение в методологию DevOps 31

4.2 Порядок выполнения работы 32

4.2.1 Подготовка к выполнению ЛР№3 ..... *33*

4.2.2 Изменение кода программы *33*

4.2.3 Запуск тестов и мониторинг *39*

4.3 Варианты заданий 40

4.4 Требования к отчёту 40

4.5 Контрольные вопросы 41

5 ЛИТЕРАТУРА **42**

ПРИЛОЖЕНИЕ А. СОГЛАШЕНИЕ О СТИЛЕ КОДИРОВАНИЯ **43**

ПРИЛОЖЕНИЕ Б. НАСТРОЙКА РАБОТЫ С ВУЗОВСКИМ РЕПОЗИТОРИЕМ КОДА **49**

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СОКРАЩЕНИЙ И ТЕРМИНОВ 52

Аннотация

Настоящее электронное учебное издание содержит указания и требования к порядку выполнения лабораторных работ. Дается теоретический материал, необходимый для выполнения работ, и предлагается пример реализации программ, аналогичных разрабатываемым на лабораторных работах. Определяются цели и объем, требования к отчетам, а также приводятся варианты заданий и примерный список контрольных вопросов для защиты выполненных заданий.

Пособие предназначено для студентов первого курса магистратуры специальности «Прикладная информатика».

Введение

Практические работы по дисциплине «Современные технологии разработки программного обеспечения» посвящены приобретению практических навыков проектирования и разработки программ на современных стандартах языка C++ при использовании гибких методик разработки программного обеспечения.

1 Общее описание практикума

Программное обеспечение (ПО) имеет естественную тенденцию к постоянному усложнению, в то время как сроки разработки имеют потребность в сокращении. Такая ситуация становится возможной благодаря развитию разнообразных приемов организации разработки (методологий), инструментальных средств, а также техник кодирования.

Перечислим основные признаки сложного ПО:

– **комплексность** – то есть, собственно, сложность, которая проявляется в большом количестве объектов, их вложенности друг в друга и связями между ними;

– **длительный жизненный цикл** – то есть необходимость не только разработать ПО, но и поддерживать его развитие на протяжении длительного времени;

– **работа в команде** – что подразумевает невозможность разработать и поддерживать сложное ПО в одиночку за приемлемое время, а также устойчивость проекта в условиях текучки команды разработки.

Особенно ярко разрешение проблемы сокращения времени разработки сложного ПО проявляется в гибких методиках разработки программного обеспечения, в которых делается особый акцент на технике построения устойчивого к изменениям кода, так как в условиях интерактивной разработки могут производиться частые и существенные уточнения требований к ПО.

Для реализации устойчивого кода используются следующие приемы:

– применение объектно-ориентированной парадигмы с осознанным выделением уровней состояния объектов и инвариантов классов [8];

– построение многоуровневой архитектуры приложения с максимальным разделением уровней [8];

– применение устойчивых к изменению техник, основанных на принципах RAII [4], SOLID [6] и других;

– использование рекомендуемых приемов (паттернов) проектирования [5];

– использование инструментальных средств, предоставляющих возможность статического анализа кода непосредственно во время его написания;

– максимальное использование принципов автодокументирования кода (см. раздел 1.3).

Помимо написания устойчивого к изменениям кода, важным при разработке сложного ПО является работа в команде с общим репозиторием исходного кода (см. раздел 1.5). Данная техника в лабораторном практикуме является рекомендуемой. В частности, дается возможность сформировать «команду» до двух человек, а также использовать вузовский репозиторий хранения исходного кода [9]. Кроме того, при выполнении лабораторных работ рекомендуется использовать автотестирование и автопостроение диаграмм классов, которые выполняются автоматически при выкладывании кода в репозиторий.

Все задания, примеры и шаблоны для выполнения лабораторных работ оформляются на языке программирования C++ с применением стандарта ISO/IEC 14882:2017 [10], который более известен в виде обозначения C++17.

1.1 Взаимосвязь элементов практикума

В таблице 1 приведены взаимосвязь элементов практикума и условие формирования отчета.

Таблица 1 – Взаимосвязь элементов практикума и условие формирования отчета

Название практикума	**Зависимость от других элементов**	**Возможность работы в «команде»**	**Форма предоставления отчета**
ЛР №1 «Описание модели, изоляция предметной области»		да	CI
ЛР №2 «Рефакторинг. Выделение сущностей, значений и служб модели»	ЛР №1	да	CI

1.2 Общие требования к отчету

Все записи в отчете должны содержаться в файле «README.md» в корне проекта. Схемы и рисунки должны быть размещены в каталоге doc или image проекта и вставлены в «README.md» (см. шаблон проекта лабораторной работы №1 по адресу https://github.com/anton-petrov/msdtm/tree/master/labs/1).

Каждый отчет должен иметь титульный лист, на котором указывается:

– наименование факультета и кафедры;

– название дисциплины;

– номер и тема лабораторной работы;

– фамилия преподавателя, ведущего занятия;

– фамилия, имя и номер группы студента;

– номер и название варианта задания;

– ссылку на проект выполненной лабораторной работы в GitHub (например: https://github.com/anton-petrov/msdtm/tree/master/labs/1).

Кроме того, отчет должен содержать:

– краткое описание решения (требования к конкретным отчетам описаны ниже);

- диаграмму классов (если требуется);
- другие диаграммы (если требуется);
- выводы.

1.3 Требование к стилю кодирования

При разработке сложного ПО большое значение имеет формирование устойчивого к изменениям кода (такой код также часто называют «адаптивным» или «безопасным»). При этом, в условиях длительного жизненного цикла проекта и работы в нём команды разработчиков, которая может меняться, особое значение имеет соглашение о стиле кодирования.

В приложении Б приводится перечень основных пунктов соглашения о стиле кодирования на языке программирования C++. Важно заметить, что пункты соглашения представляют собой рекомендации, которые могут быть нарушены по следующим причинам:

- нарушение улучшает читаемость кода,
- нарушение необходимо для достижения высокой производительности программы.
- Если какой-либо пункт соглашения был нарушен, необходимо в коде оставить комментарий, объясняющий причину такого нарушения.

1.4 Установка и настройка GNU/Linux Ubuntu 18.04 LTS

При формировании отчетов к лабораторным работам рекомендуется использовать репозиторий исходного кода GitLab или GitHub. При использовании репозитория могут быть применены средства непрерывной интеграции. Эти средства исполняются в среде операционной системы GNU/Linux, поэтому рекомендуется записывать и отлаживать программу в среде той же операционной системы.

В качестве дистрибутива операционной системы GNU/Linux рекомендуется взять Ubuntu версии 18.04 LTS (или более новой) в силу наибольшей совместимости со средой исполнения средств непрерывной интеграции.

В таблице 2 приведен перечень операций по установке операционной системы GNU/Linux Ubuntu 18.04 LTS.

Таблица 2 – Операции по установке операционной системы GNU/Linux Ubuntu 18.04 LTS

№№	Операция	Пояснения
1	Выбор способа работы с операционной системой	Операционную систему GNU/Linux Ubuntu 18.04 LTS можно установить следующими способами: – на компьютер (ноутбук) как основную ОС; – на компьютер (ноутбук) совместно с другой ОС, например, с MS Windows; – в среду виртуализации, например, VirtualBox.
2	Установка GNU/Linux Ubuntu 18.04 LTS в соответствии с выбранным способом	Каким образом устанавливать GNU/Linux Ubuntu 18.04 LTS можно прочитать/посмотреть на различных ресурсах Интернет.
3	Установка пакетов разработки	Для установки компилятора C++ нужно выполнить в терминале следующую команду: `sudo apt install build-essential`
4	Установка среды разработки	В качестве среды разработки рекомендуется использовать Visual Studio Code. Это можно сделать с сайта продукта (https://code.visualstudio.com). В качестве среды разработки можно также использовать Qt Creator, установив пакет «Open Source» с сайта продукта (https://www.qt.io/download). При этом нужно учесть следующие особенности: – после скачивания файла установщика, необходимо изменить его свойства, добавив разрешение на исполнение; – после установки Qt в терминале нужно выполнить следующую команду[1]: `sudo apt-get install libgl1-mesa-dev`

[1] Данная команда не обязательна для работы с терминальными приложениями без использования библиотеки Qt, но если возникнет необходимость работать с библиотекой Qt, то без выполнения этой команды, при компиляции программы будет выдаваться сообщение об ошибке.

1.5 Использование репозитория исходного кода

Системы управления версиями Git является распределенной, то есть:

– предоставляет каждому разработчику локальную историю разработки,

– не требует захватывать файлы для выполнения изменений.

Также Git позволяет выполнять быстрое и удобное разделение и слияние версий ПО. В настоящее время Git является одной из наиболее распространенных систем управления версиями.

Система GitLab является Web-приложением и системой управления репозиториями кода для Git. Она предоставляет следующие возможности:

– организацию публичных и приватных репозиториев;

– управление правами, группами;

– импорт проектов, в том числе с GitHub;

– работу с вики-страницами для проекта;

– API для внешнего управления;

– работу с доской идей и задач;

– настройку лейблов, вех, шаблонов;

– возможности поиска;

– комментирование действий и изменений, объединение веток;

– реализацию непрерывной интеграции и непрерывного развертывания (CI/CD) с поддержкой DevOps;

– отслеживание изменений и прогресса;

– отслеживание времени;

– и т.д.

– В приложении В приводится последовательность действий для установки и настройки системы Git, а также подключения к GitLab и настройки работы с проектами в этой системе.

2 Лабораторная работа №1. Реализация приложения в многоуровневой архитектуре с использованием удалённого репозитория кода

Цель работы: приобрести навыки разработки приложения в многоуровневой архитектуре с использованием системы управления версиями Git и удаленного репозитория GitLab.

Объем работы: 5 часов.

Студенты должны проанализировать предложенный шаблон решения многоуровневой архитектуры, выполнить реализацию задачи в виде программы и сохранить результаты работы в удаленном репозитории GitLab с использованием системы управления версиями Git.

2.1 Изоляция предметной области

Предметно-ориентированное проектирование (Domain-driven design, **DDD**) – это набор принципов и схем, направленных на создание оптимальных систем объектов. DDD сводится к созданию программных абстракций, которые называются моделями предметных областей и обладает рядом плюсов:

- позволяет автоматизировать незнакомые разработчикам предметные области;
- позволяет вести разработку итерационно, постепенно усложняя ПО;
- позволяет значительно ускорить разработку сложного ПО.

DDD в свое время собрало наиболее успешные принципы и техники проектирования программного обеспечения в методологиях гибкой разработки (Agile) и стало основой для многих других техник проектирования.

Приведем несколько ключевых определений:

Область (Domain) — предметная область, к которой применяется разрабатываемое программное обеспечение.

Язык описания — используется для единого описания модели предметной области.

Модель (Model) — описывает конкретную предметную область или её часть, является базой для автоматизации.

Описание модели предметной области в DDD сводится к построению UML-диаграмм и затем реализации их в коде программы. Код модели должен отражать модель предметной области. Изменение модели влечет изменение кода и, наоборот, изменение кода означает изменение модели.

Изоляция предметной области необходима для отделения кода модели от вспомогательных конструкций, что позволяет выполнять гибкую и независимую модификацию участков кода, отвечающих за принципиально разный функционал (см. SRP SOLID или ШП

Functional design), вплоть до полной замены одних модулей без радикального изменения других.

Изоляция предметной области как правило выполняется с использованием многоуровневой архитектуры, которая имеет следующий канонический состав уровней [8]:

1) **Интерфейс пользователя** (уровень представления) – UI. Отвечает за вывод информации пользователю и интерпретацию команд пользователя.
2) **Операционный уровень** (уровень прикладных операций, уровень приложения) – AL. Определяет задачи, связанные с конкретным действием в UI (команда пользователя или потребность в информации) и распределяет их между объектами предметной области. Не хранит состояний объектов предметной области. Может играть интегрирующую роль – взаимодействовать с операционными уровнями других систем. Наиболее близкий ШП: Fasade.
3) **Уровень предметной области** (уровень модели, уровень бизнес-логики) – DL. Отвечает за представление понятий прикладной предметной области, рабочие состояния, бизнес-регламенты (поведение модели). Этот уровень является главной, алгоритмической частью программы.
4) **Инфраструктурный уровень** (уровень доступа к данным) – IL. Слой технических сервисов. Обеспечивает техническую поддержку для верхних уровней: передачу сообщений на операционном уровне, непрерывность существования объектов на уровне модели (хранение, транзакционность и т.д.), службы передачи сообщений, почтовые службы и т.д.

Каждый уровень зависит только от нижележащего слоя и может существовать без вышерасположенных слоёв (см. рисунок 1).

Для связи с верхними уровнями используются:

- обратные вызовы (callback);
- ШП Observer;
- стандартные архитектурные шаблоны:
  - Model-View-Controller (MVC),
  - Model-View-Presenter,
  - Naked objects,
  - и др.

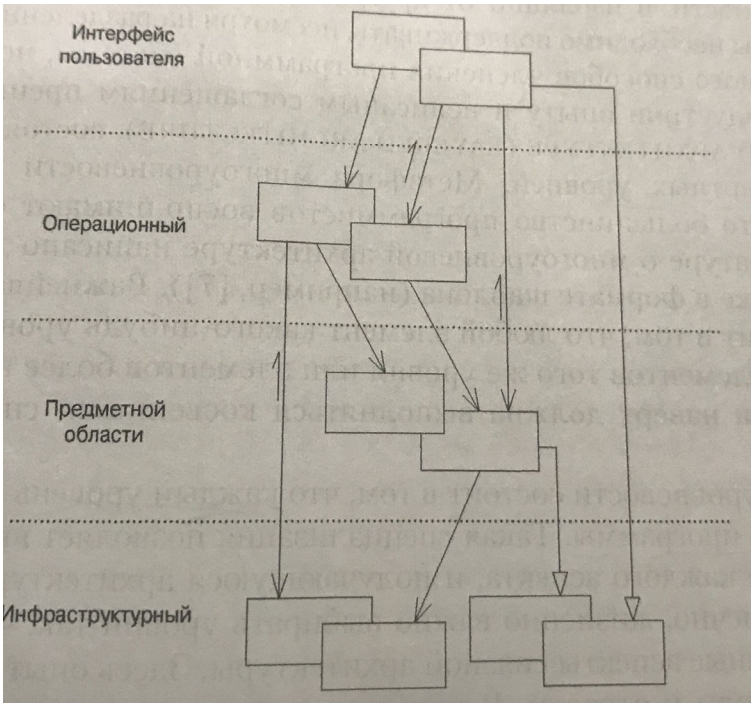

Рисунок 1 – Условная схема взаимодействия объектов программы, расположенных на разных уровнях

2.2 Порядок выполнения работы

Перед выполнением лабораторной работы необходимо получить у преподавателя номер задачи/карточки, которые приведены в таблице 3, а также вариант задания (см. 2.3 «Варианты заданий»).

Таблица 3 – Варианты заданий

№№	**Наименование задачи**	**Карточка задания**	**Описание карточки**
1.	Задача: «Наблюдаемые суда»	Карточка: «Морское судно»	Карточка должна содержать следующие поля: – наименование судна, – сухой тоннаж, – полный тоннаж, – тип (сухогруз, танкер, круизный лайнер).

№№	Наименование задачи	Карточка задания	Описание карточки
2.	Задача: «Война в долине теней»	Карточка: «Воин Долины теней»	Карточка должна содержать следующие поля: – фракция (перечисляемый тип), – сила удара, – защита, – здоровье, – ловкость, – уклонение, – тип боя (ближний, дальний).
3.	Задача: «Автозавод»	Карточка: «Модель автомобиля»	Карточка должна содержать следующие поля: – наименование модели, – номер сборочного конвейера, – код краски (перечисляемый тип), – код комплектации (перечисляемый тип)
4.	Задача: «Интернет-магазин»	Карточка: «Товар»	Карточка должна содержать следующие поля: – наименование товара, – стоимость товара, – дата занесения в БД, – наличие товара.
5.	Задача: «Картинная галерея»	Карточка: «Картина»	Карточка должна содержать следующие поля: – наименование картины, – автор, – год создания, – живописная техника (масло, графика, другое).

№№	Наименование задачи	Карточка задания	Описание карточки
6.	Задача: «Воздушная обстановка»	Карточка: «Позиция борта»	Карточка должна содержать следующие поля: – номер борта, – тип воздушного судна (пассажирский, грузовой, военный), – размер воздушного судна (большой, средний, малый), – координаты (x, y, z).
7.	Задача: «Клубная лига»	Карточка: «Футбольный клуб»	Карточка должна содержать следующие поля: – название клуба, – текущее место в лиге, – количество запасных, – ФИО тренера.
8.	Задача: «Экологическая обстановка»	Карточка: «Экологическая карточка»	Карточка должна содержать следующие поля: – название населенного пункта, – код в справочнике ОКАТО, – количество жителей, – уровень загрязнения (высокий, средний, низкий).
9.	Задача: «Карта галактики»	Карточка: «Звездная система»	Карточка должна содержать следующие поля: – номер по каталогу, – наименование (если есть), – дальность от Земли, – масса, – класс (O, B, A, F, G, K, M).
10.	Задача: «Коллекция монет»	Карточка: «Монета»	Карточка должна содержать следующие поля: – специальное наименование (если есть), – тип металла (перечисляемый тип), – наименование валюты, – количество единиц валюты, – количество монет в коллекции.

№№	Наименование задачи	Карточка задания	Описание карточки
11.	Задача: «Музыкальная коллекция»	Карточка: «Музыкальный трек»	Карточка должна содержать следующие поля: – название трека, – автор, – исполнитель, – альбом, – длительность, – оценка (от 0 до 5).
12.	Задача: «Магазин музыкальных инструментов»	Карточка: «Инструмент»	Карточка должна содержать следующие поля: – название инструмента, – производитель, – модель, – стоимость, – наличие.
13.	Задача: «Магазин автозапчастей»	Карточка: «Автозапчасть»	Карточка должна содержать следующие поля: – название автозапчасти, – марка автомобиля, – модель автомобиля, – стоимость, – наличие.
14.	Задача: «Выборы в Муниципалитет»	Карточка: «Кандидат»	Карточка должна содержать следующие поля: – ФИО кандидата, – возраст, – среднегодовой доход, – партия, – количество проголосовавших.
15.	Задача: «Букмекерская компания»	Карточка: «Ставка»	Карточка должна содержать следующие поля: – номер ставки, – номер клиента, – коэффициент, – сумма ставки.

№№	Наименование задачи	Карточка задания	Описание карточки
16.	Задача: «Ресторан быстрого питания»	Карточка: «Блюдо»	Карточка должна содержать следующие поля: – название блюда, – стоимость, – количество калорий, – время приготовления, – популярность (от 0 до 5).
17.	Задача: «Курьерская служба»	Карточка: «Заказ»	Карточка должна содержать следующие поля: – номер заказа, – ФИО курьера, – название товара, – адрес доставки, – стоимость доставки.
18.	Задача: «Магазин приложений»	Карточка: «Приложение»	Карточка должна содержать следующие поля: – название приложения, – категория, – стоимость, – размер, – количество установок, – оценка (от 0 до 5).
19.	Задача: «Онлайн-школа программирования»	Карточка: «Курс»	Карточка должна содержать следующие поля: – название курса, – продолжительность, – язык программирования, – сложность курса, – стоимость обучения.
20.	Задача: «Каталог библиотеки»	Карточка: «Книга»	Карточка должна содержать следующие поля: – название книги, – автор, – издательство, – год издания, – жанр.

№№	Наименование задачи	Карточка задания	Описание карточки
21.	Задача: «Электронный журнал»	Карточка: «Студент»	Карточка должна содержать следующие поля: – ФИО студента – ID студента – год поступления – кафедра – учёная степень (перечисляемый тип) – рейтинг (от 0 до 100)
22.	Задача: «Банк»	Карточка: «Транзакция»	Карточка должна содержать следующие поля: – получатель – отправитель – сумма – код валюты – код транзакции – дата транзакции
23.	Задача: «Авиа-рейс»	Карточка: «Бронь»	Карточка должна содержать следующие поля: – ФИО пассажира (если есть) – номер места – номер брони – статус (занято/свободно) – стоимость – тип места (эконом/бизнес/комфорт)
24.	Задача: «Онлайн-кинотеатр»	Карточка: «Фильм»	Карточка должна содержать следующие поля: – название фильма – режиссёр – жанр – год – рейтинг (от 0 до 5) – длительность

№№	Наименование задачи	Карточка задания	Описание карточки
25.	Задача: «Музыкальный фестиваль»	Карточка: «Группа»	Карточка должна содержать следующие поля: – название группы – жанр – порядок выступления – время (начало, конец)

Суть лабораторной работы заключается в разделении исходного кода на модули C++ в соответствии со слоями многоуровневой архитектуры:

1) Модуль «**l1_UserInterface.cpp**» (без соответствующего заголовочного файла), отвечающий за реализацию интерфейса с пользователем.
2) Модуль «**l2_ApplicationLayer.cpp**» (заголовочный файл «**l2_ApplicationLayer.h**»), реализующий операционный уровень.
3) Модуль «**l3_DomainLayer.cpp**» (заголовочный файл «**l3_DomainLayer.h**»), реализующий уровень модели предметной области.
4) Модуль «**l4_InfrastructureLayer.cpp**» (заголовочный файл «**l4_InfrastructureLayer.h**»), реализующий инфраструктурный уровень.

Модули C++ должны располагаться в каталоге «src», а их заголовочные файлы (если есть) – в каталоге «include/hw».

Проект должен содержать следующие каталоги:

– **bin** – для хранения исполняемого файла (не сохраняется в репозитории, создается командным файлом «configure»);

– **build** – для хранения объектных файлов модулей программы (не сохраняется в репозитории, создается командным файлом «configure»);

– **doc** – для хранения конфигурационного файла «Doxyfile.cfg» и временных файлов сгенерированной документации, а также изображений;

– **include/hw** – для хранения заголовочных файлов;

– **src** – для хранения кода модулей на C++;

– **test** – для хранения автотестов (не изменяется по сравнению с выполненой ДР).

В корневом каталоге проекта должны находиться следующие файлы:

– **.gitignore** – список исключений для git;

– **.gitlab-ci.yml** – описание сценария CI;

– **configure** – командный файл, выполняющий первоначальную настройку проекта;

– **documentation** – командный файл, формирующий документацию по исходному коду проекта в каталоге doc/html;

– **hw.pro** – проектный файл для Qt Creator (не обязательный);

- **LICENSE.MIT** – лицензия MIT на английском языке;
- **LICENSE-RUS.MIT** – лицензия MIT на русском языке;
- **Makefile** – управляющий файл для сборки проекта с использованием утилиты make;
- **README.md** – файл описания проекта.

Заготовку проекта можно взять из репозитория по адресу https://github.com/anton-petrov/msdtm/tree/master/labs.

Для выполнения сборки проекта используется утилита make, которую нужно установить на компьютер, если она не установлена. Для выполнения сборки проекта нужно выполнить команду make из рабочего каталога проекта. Для выполнения автотестов нужно выполнить команды test/test01 и test/test02.

Диаграммы классов, описывающие модель предметной области, можно взять из генерируемой документации.

Также необходимо взять вариант задания и нарисовать диаграмму последовательности выполнения команды, изобразив прохождение выполнения команды между программными объектами, расположенными в соответствующих слоях многоуровневой архитектуры. Скан этого рисунка нужно разместить в файле описания проекта «README.md». Образец можно посмотреть в заготовке проекта.

2.3 Варианты заданий

Варианты заданий для выполнения лабораторной работы №3 приведены в таблице 4.

Таблица 4 – Варианты заданий для выполнения лабораторной работы №1

№№	Задание
1.	Постройте диаграмму последовательности выполнения команды **`save`**. Диаграмма должна показывать прохождение выполнения команды между программными объектами, расположенными в соответствующих слоях многоуровневой архитектуры.
2.	Постройте диаграмму последовательности выполнения команды **`clean`**. Диаграмма должна показывать прохождение выполнения команды между программными объектами, расположенными в соответствующих слоях многоуровневой архитектуры.
3.	Постройте диаграмму последовательности выполнения команды **`add`**. Диаграмма должна показывать прохождение выполнения команды между программными объектами, расположенными в соответствующих слоях многоуровневой архитектуры.
4.	Постройте диаграмму последовательности выполнения команды **`remove`**. Диаграмма должна показывать прохождение выполнения команды между программными объектами, расположенными в соответствующих слоях многоуровневой архитектуры.

№№	Задание
5.	Постройте диаграмму последовательности выполнения команды **`update`**. Диаграмма должна показывать прохождение выполнения команды между программными объектами, расположенными в соответствующих слоях многоуровневой архитектуры.
6.	Постройте диаграмму последовательности выполнения команды **`view`**. Диаграмма должна показывать прохождение выполнения команды между программными объектами, расположенными в соответствующих слоях многоуровневой архитектуры.

2.4 Требования к отчету

Отчет должен содержать:

- номер и текст задачи/карточки;
- номер и текст задания для ЛР№1;
- диаграммы классов проекта;
- диаграмму последовательности по полученному заданию;
- выводы.

Лабораторная работа №1 должна быть сохранена в репозитории с использованием CI. В этом случае отчёт должен быть выполнен в файле описания проекта «README.md». Предоставлять бумажный отчет не нужно.

2.5 Контрольные вопросы

После выполнения данной контрольной работы нужно уметь ответить на следующие контрольные вопросы:

1) Дайте определение изоляции предметной области?
2) Каким образом выполняется изоляция предметной области?
3) Какие преимущества достигаются при изоляции предметной области?
4) Назовите слои многоуровневой архитектуры.
5) Опишите назначение уровня пользовательского интерфейса.
6) Опишите назначение операционного уровня.
7) Опишите назначение уровня предметной области.
8) Опишите назначение инфраструктурного уровня.

3 Лабораторная работа №2. Реализация модели предметной области с использованием техники непрерывной интеграции

Цель работы: приобрести навыки построения модели предметной области и разработки приложения, работая в проектной команде с использованием техники непрерывной интеграции GitLab.

Объем работы: 5 часов.

Студенты должны проанализировать задачу, построить модель предметной области и, работая в команде, выполнить реализацию решения в виде программы, а также сохранить результаты работы в удаленном репозитории GitLab с использованием системы управления версиями Git и с применением техники непрерывной интеграции.

3.1 MDD, структура модели, углубляющий рефакторинг

Техника DDD чаще всего и наиболее эффективно применяется при гибких методологиях разработки, когда не только осознанно допускается, но и прямо декларируется несоответствие модели и предметной области. Это понимание основано на том, что, во-первых, в процессе автоматизации незнакомой предметной области длительное время невозможно точно отразить предметную область в модели – она просто неизвестна. Это отражение приходится делать постепенно, итерационно, постоянно консультируясь с представителями заказчика, показывая им и обсуждая результаты каждого цикла.

Во-вторых, даже если предметная область в целом знакома, но есть другие риски, связанные с успехом автоматизации, целесообразно вести разработку опять же итерационно, в первую очередь реализуя главный функционал, наиболее востребованный заказчиком.

Такой подход определяет важность соблюдения соответствия модели и кода программы, когда стремятся минимизировать временной лаг между разработкой модели, реализуемой в данной итерации функционала и кодом программы, что позволяет утверждать, что код программы и является моделью предметной области. Этот подход закрепляется в технике проектирования по модели, которая является основой DDD.

Проектирование по модели (model-driven design, **MDD**) – проектирование архитектуры, при котором соблюдается максимально точное соответствие между некоторым подмножеством элементов программы и элементами модели или, иными словами, процесс совместной разработки модели и её программной реализации с сохранением такого соответствия между ними [8].

Построение модели обычно выполняется в следующей последовательности:

1) Выделение **понятий** предметной области, участвующих в реализуемом функционале (применяется обычная техника абстрагирования ООП).

2) Построение **ассоциаций** между понятиями предметной области и их максимальное упрощение. Упрощение ассоциаций сводится к следующим правилам:
   а) сведение связей к неравноправным (направленным);
   б) определение кратности связей, устранение связей «многие-ко-многим»;
   в) минимизация кратности за счёт ограничений ассоциации (квалификатором ассоциаций).
3) Выделение **сущностных понятий** предметной области – объектов (понятий) предметной области, характеризующихся непрерывностью и индивидуальностью (уникальностью) существования.
4) Выделение **объектов-значений** – понятий, которые представляют описательный аспект предметной области и не имеют индивидуального существования, собственной идентичности.
5) Выделение **служб** – операций или групп операций, предлагаемых в модели в виде обособленного интерфейса, который в отличие от сущности или объекта-значения не имеет никакого состояния.
6) Разделение модели на **модули** – устоявшихся элементов архитектуры программы.
7) Объединение объектов модели (и программы) в **агрегаты** – совокупность взаимосвязанных объектов, которые мы воспринимаем как единое целое с точки зрения изменения данных. Понятие агрегата очень важно в проектировании по модели, так как выделяет из набора понятий корневую (главную) сущность, которая позволяет управлять транзакционностью, контролем инварианта агрегата, выполнять создание целостной совокупности объектов и хранением её состояния. При работе с агрегатами следует придерживаться следующих правил:
   а) корневой объект-сущность имеет глобальную идентичность и несёт полную ответственность за проверку инвариантов;
   б) некоренные объекты-сущности имеют локальную идентичность – они уникальны только в границах агрегата;
   в) нигде за пределами агрегата не может храниться ссылка на что-либо внутри него (только временные ссылки и значения);
   г) значит, только корневые объекты агрегатов можно непосредственно получать по запросам из БД, все остальные объекты разрешается извлекать по цепочке связей;
   д) объекты внутри агрегата могут хранить ссылки на корневые объекты других агрегатов;
   е) операция удаления должна ликвидировать всё, что находится в границах агрегата;
   ж) как только вносится изменение в любой объект агрегата, следует сразу удовлетворять все инварианты этого агрегата.

Перечисленные действия обычно выполняются на каждой итерации разработки, когда изменяется модель предметной области. При этом часто сначала выполняется рефакторинг существующей структуры модели, после которого прогоняются регрессионные тесты, а затем осуществляется добавление или изменение функциональности.

Рефакторинг – это такая реструктуризация программы, в результате которой не изменяются её функциональные возможности [8]. Рефакторинг выполняется, если необходимо сделать модель объектов проще, понятнее или как предварительный этап перед добавлением новой функциональности. Важным является то, что после рефакторинга регрессионные тесты должны выполняться без изменений.

При **углубляющем рефакторинге** модель изменяется с учётом дополнительной функциональности при этом стараются изменить структуру так, чтобы она не требовала изменений автотестов.

3.2 Порядок выполнения работы

Предыдущий практикум (лабораторная работа №1) был посвящён выделению и изоляции предметной области, то есть модели, и это позволит упростить и ускорить разработку кода модели. При выполнении лабораторной работы №2 необходимо взять за основу программу, составленную при выполнении лабораторной работы №1.

Далее нужно самостоятельно придумать или взять вариант, приведённый в подразделе 3.3. Текст придуманного или взятого условия задания нужно отразить в отчёте.

Далее нужно построить диаграмму классов (объектов модели предметной области), выделить на ней сущности, объекты-значения и службы, а также показать агрегат, обведя объекты, которые в него входят, и выделить главный объект-сущность, который должен выступать вовне из обведённых границ агрегата. Диаграмму необходимо нарисовать на листе, который затем отсканировать и вставить в файл README.md, см. рисунок 2.

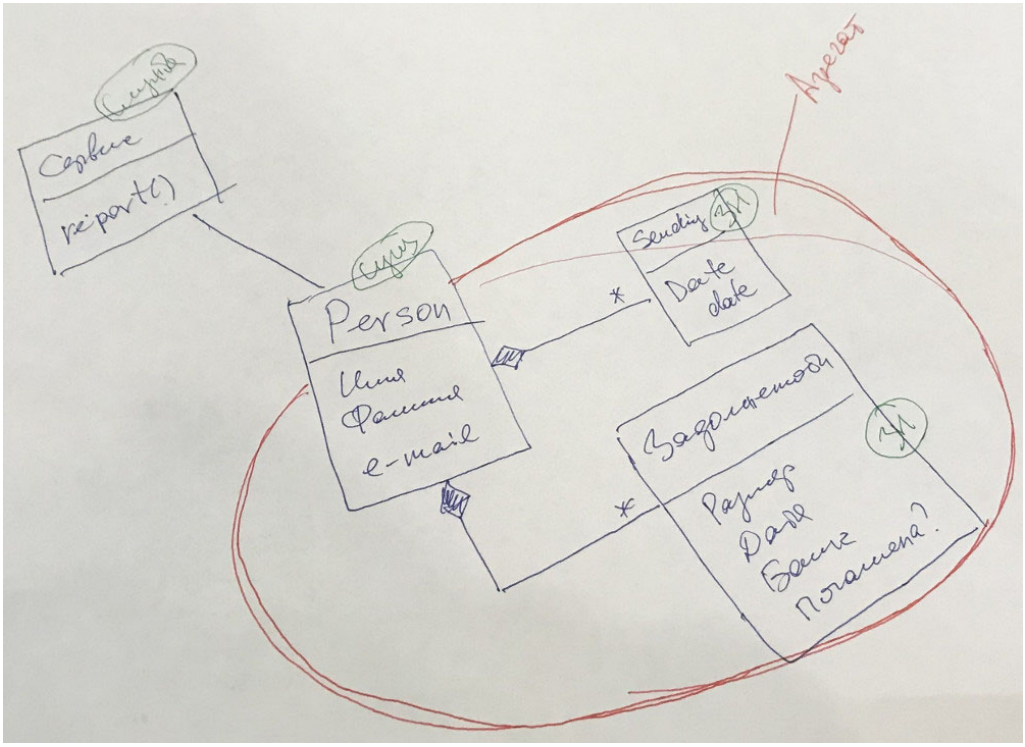

Рисунок 2 – Пример диаграммы классов модели с помеченными сущностями, объектами-

значениями, службами и выделенным агрегатом с главным объектом-сущностью.

Затем на базе выполненной ранее лабораторной работы №1 нужно реализовать:

– рефакторинг, отражающий изменения в модели,
– убедиться, что он не повлиял на старую функциональность,
– добавить новую функциональность, доработать автотесты и
– сохранить в репозитории.

Код образца выполнения ЛР№2 можно посмотреть в репозитории или взять у преподавателя.

3.3 Варианты заданий

Варианты заданий для выполнения лабораторной работы №2 можно придумать самостоятельно на базе выданного ранее варианта задания (ваш вариант должен предполагать невырожденный случай агрегата) или взять в таблице 5.

Таблица 5 – Варианты заданий для выполнения лабораторной работы №2

№№ ДЗ	Наименование задачи / карточки	Задание
1.	«Наблюдаемые суда» / «Морское судно»	Необходимо определить три морских судна, находящихся на минимальном прямом расстоянии от другого судна, которое терпит бедствие и посылает сигналы SOS.
2.	«Война в долине теней» / «Воин Долины теней»	Ваш отряд встречает ужасного тролля, у которого здоровье равно сумме здоровья всех ваших бойцов, сила удара и защита равны силе удара и защите самого сильного и защищённого из бойцов, соответственно, а все остальные характеристики равны минимальной из всего вашего отряда. Как победить тролля с минимальными потерями? В битве урон здоровью после удара вычисляется по формуле: (1) $Урон_1 = \lvert Защита_1 + Уклонение_1 – Сила\_удара_2 – Ловкость_2 \rvert$
3.	«Автозавод» / «Модель автомобиля»	Предположим, что на вашем заводе 10 конвейеров. Каждому из них требуется плановый ремонт по заданному расписанию. Определите, требуется ли перенастраивать другой конвейер для выпуска продукции ремонтируемого, используя информацию об излишках продукции на заводском складе.

№№ ДЗ	Наименование задачи / карточки	Задание
4.	«Интернет-магазин» / «Товар»	Составьте отчёт для курьера вашего интернет-магазина с информацией об адресах доставки товаров, основываясь на данных оплаченных, но ещё не развезённых заказах.
5.	«Картинная галерея» / «Картина»	Часть ваших картин находится на экспозиции, часть хранится на складе. Нужно подготовить отчёт по картинам определённого автора, которые можно временно передать на выставку в другом музее, не нарушая существующую экспозицию.
6.	«Воздушная обстановка» / «Позиция борта»	Ваш небольшой аэродром может принимать только малый размер воздушных объектов, а также только гражданских. Необходимо регистрировать все запросы на посадку, а давать её только разрешённым суднам. В случае запроса на экстренную посадку можно давать разрешение средним и военным суднам, при этом остальные запросы отклоняются. Составьте систему автоматического управления разрешениями на посадку (сформируйте отчет разрешений на основании текущей воздушной обстановки).
7.	«Клубная лига» / «Футбольный клуб»	Необходимо составить расписание матчей чемпионата по правилам выбывания на следующий сезон, в котором принимают участие 16 лучших команд. При этом нужно учесть распределение первых 4 по рейтингу команд так, чтобы все они потенциально могли занять призовые места.
8.	«Экологическая обстановка» / «Экологическая карточка»	Необходимо определить степень увеличения загрязнения населённых пунктов, которое может произойти из-за выброса ядовитых веществ на предприятии в одном из них. При этом увеличение загрязнения рассчитывается по степени удалённости населённого пункта от эпицентра (обратно квадрату расстояния). Получите отчёт о 5 наиболее загрязнённых населённых пунктах.
9.	«Карта галактики» / «Звездная система»	Человечество отправило автоматические станции на разведку к 5 ближайшим звёздным системам. Определите, когда мы можем получить сигналы от них о достижении цели, если все они движутся с разными скоростями.
10.	«Коллекция монет» / «Монета»	Произошло ограбление музея с редкими монетами. Страховой компании необходимо составить отчёт по страховой выплате музею, если известно, что часть экспозиции сохранилась, а условная стоимость экспонатов зафиксирована в специальной таблице.

№№ ДЗ	Наименование задачи / карточки	Задание
11.	«Музыкальная коллекция» / «Музыкальный трек»	На основании предпочтений (рейтинга), устанавливаемого для треков пользователями вашего проигрывателя, вы должны предложить индивидуальные рекомендации конкретному пользователю.
12.	«Магазин музыкальных инструментов» / «Инструмент»	На основании информации о сделанных и выполненных заказах, а также наличии инструмента на складе, необходимо сделать заказ на пополнение склада с инструментами.
13.	«Магазин автозапчастей» / «Автозапчасть»	Необходимо сформировать ежемесячный отчёт о просроченных товарах для списания. Этот отчёт необходимо сохранять на постоянной основе, т.к. эта информация будет учитываться в будущем.
14.	«Выборы в Муниципалитет» / «Кандидат»	Необходимо построить отчёт по распределению рейтинга партий для определения мест в Муниципалитет на основании результатов выборов их кандидатов в округах. Нужно учесть, что по какому-то округу выборы могут быть признаны недействительными и все результаты по этому округу учитывать не нужно.
15.	«Букмекерская компания» / «Ставка»	Необходимо построить отчёт с перечнем причитающихся премий для победителей в очередном розыгрыше. Премия вычисляется по следующей формуле: (1) $P_i = S_r * V_i / S_c$ (2) $S_r = S_c – S_{\%}$, где: P_i – премия i-го игрока-победителя; S_r – сумма всех премий; V_i – ставка выигравшего игрока; S_c – сумма всех ставок; $S_{\%}$ – часть общей суммы, отходящей букмекерской конторе.
16.	«Ресторан быстрого питания» / «Блюдо»	Необходимо посчитать калорийность блюда на основании калорийности ингредиентов.
17.	«Курьерская служба» / «Заказ»	Составьте отчёт для курьера вашего интернет-магазина с информацией об адресах доставки товаров, основываясь на данных оплаченных, но ещё не развезённых заказах.
18.	«Магазин приложений» / «Приложение»	На основании подготовленного пользователем заказа, а также истории выполненных им заказов необходимо подготовить перечень рекомендаций для дополнительных товаров «с этим покупают следующее: …».

№№ ДЗ	Наименование задачи / карточки	Задание
19.	«Онлайн-школа программирования» / «Курс»	Необходимо составить отчёт для обзвона (или для рассылки почтового сообщения) пользователей, которые сформировали заказ, но ещё не выполнили оплату.
20.	«Каталог библиотеки» / «Книга»	Необходимо сформировать отчёт по книгам, которые находятся на руках у читателей и закончились в хранилище, чтобы заказать дополнительные экземпляры. Читатель может взять несколько книг.
21.	«Электронный журнал» / «Студент»	Нужно составить отчёт о премировании преподавателей, которые читают курсы с наиболее высоким средними баллами. Отчет должен выводить преподавателей по убыванию суммарного рейтинга. В отношении первых пяти преподавателей в этом списке будет запущена процедура расследования дела о халатности в простановке оценок.
22.	«Банк» / «Транзакция»	Необходимо выполнить проверку возможности выполнения транзакции: а) у отправителя и получателя счета должны быть разблокированы; б) у отправителя тип счёта должен допускать возможность списания средств, у получателя – возможность начисления средств; в) у отправителя на счёте должна быть сумма не меньше снимаемой для перевода. Необходимо хранить всю историю транзакций: и успешных, и нет.
23.	«Авиа-рейс» / «Бронь»	Необходимо сформировать отчёт по рейсам (№ рейса, откуда, куда) с максимальным количеством просроченных броней.
24.	«Онлайн-кинотеатр» / «Фильм»	На основании просмотренных пользователем фильмов и поставленных им рейтингов необходимо подготовить перечень рекомендаций фильмов для просмотра.
25.	«Музыкальный фестиваль» / «Группа»	Каждый раз фестиваль тяжёлого рока длится несколько дней. В день выступают 2-4 группы. Организатор фестиваля хочет получить отчёт о том какие группы выступали в день, когда было продано больше всего пива, чтобы пригласить эти группы в следующий раз.

3.4 Требования к отчету

Отчет должен содержать:

- номер и текст задачи/карточки;
- номер и текст задания для ЛР№2;

– диаграммы классов модели с выделенными на ней сущностями, объектами-значениями, службами и агрегатом;

– выводы.

Заготовку проекта можно взять из репозитория по адресу https://github.com/anton-petrov/msdtm/tree/master/labs.

Лабораторная работа №2 должна быть сохранена в репозитории с использованием CI. Отчёт должен быть выполнен в файле описания проекта «README.md». Предоставлять бумажный отчет не нужно.

3.5 Контрольные вопросы

После выполнения данной контрольной работы нужно уметь ответить на следующие контрольные вопросы:

1) Дайте определение понятия «проектирования по модели» (model-driven design, MDD).
2) Дайте определение понятия «сущность».
3) Дайте определение понятия «объект-значение».
4) Дайте определение понятия «служба».
5) Дайте определение понятия «модуль».
6) Дайте определение понятия «агрегат».
7) Дайте определение понятия «рефакторинг».
8) Почему при проектировании по модели стараются сократить время между формированием модели (созданием диаграммы классов модели) и кодированием модели?
9) Зачем нужен рефакторинг?
10) Когда выполняется рефакторинг?

4 Лабораторная работа №3. Разработка простого DevOps-приложения

Цель работы: приобрести навыки разработки DevOps.

Объем работы: 5 часов.

4.1 Введение в методологию DevOps

DevOps (акроним от англ. development и operations) – методология активного взаимодействия специалистов по разработке со специалистами по информационно-технологическому обслуживанию и взаимная интеграция их рабочих процессов друг в друга для обеспечения качества продукта. Предназначена для эффективной организации создания и обновления программных продуктов и услуг. Основана на идее тесной взаимозависимости создания продукта и эксплуатации программного обеспечения, которая прививается команде как культура создания продукта.

Методологию DevOps можно считать развитием идей гибкой разработки (Agile) в крупных проектах с большим количеством пользователей. Как правило, это интернет-проекты, в которых приложение предоставляет набор сервисов. Примерами таких приложений могут быть:

– музыкальное интернет-приложение, предоставляющее возможность онлайн-проигрывания музыкальных треков, а также сопутствующие функции (поиск, коллекции и т.д.);

– службы онлайн-оформления документов, как и другие онлайн-услуги программы «электронное правительство»;

– интернет-приложение любой автоматизации предприятия для удалённой работы, как сотрудников, так и его клиентов;

– и многие другие.

Цели DevOps охватывают весь процесс поставки программного обеспечения. Они включают:

1) сокращение времени для выхода на рынок;
2) снижение частоты отказов новых релизов;
3) сокращение времени выполнения исправлений;
4) уменьшение количества времени на восстановления (в случае сбоя новой версии или иного отключения текущей системы).

Чтобы эффективно использовать DevOps, прикладные программы должны соответствовать набору архитектурно значимых требований, таких как: возможность развертывания, изменяемость, тестируемость и возможности мониторинга.

Хотя в принципе можно использовать DevOps с любым архитектурным стилем, архитектурный стиль микросервисов является стандартом для построения постоянно развернутых систем. Поскольку размер каждой услуги невелик, он позволяет создавать архитектуру отдельного сервиса посредством непрерывного рефакторинга, что уменьшает необходимость в большом предварительном дизайне и позволяет выпускать программное обеспечение на ранней стадии непрерывно.

Таким образом, чаще всего можно разделить DevOps на следующие тесно связанные составные части:

1) идеи гибкой разработки (Agile);
2) архитектуру микросервисов и связанные с ней:
   а) предметно-ориентированное проектирование (Domain Driven Design),
   б) непрерывную интеграцию и развертывание,
   в) техники автоматизации тестирования,
   г) журналирование и мониторинг;
3) строгие правила поддержки пользователей не только с точки зрения сопровождения ПО, но и с точки зрения всей инфраструктуры проекта на базе таких стандартов, как ITIL (IT Infrastructure Library – библиотека инфраструктуры информационных технологий) или более современных.

4.2 Порядок выполнения работы

Важно отметить, что первая и вторая лабораторные работы закрепляли практический аспект первых трёх подпунктов в части архитектуры микросервисов. Задачей лабораторной работы №3 является получение практики в обеспечении мониторинга в рамках одного микросервиса, выполненного с разделением слоёв. Для этого необходимо выполнить ряд изменений результата лабораторной работы №2, а именно:

1) оформить работу приложения в виде имитации работы микросервиса, для чего необходимо:
   а) заменить прекращение работы программы в случае нарушения инварианта, генерацией исключения с последующей обработкой и журналированием,
   б) оформить генерацию номера сессии работы микросервиса с последующим отражением в журнале,
   в) подсчёт времени работы сессии с последующим отражением в журнале;
2) предусмотреть имитацию использования микросервиса с обработкой полученного журнала и вывода информации в виде графиков.

При выполнении лабораторной работы рекомендуется использовать шаблонный пример реализации ЛР№3 на портале удалённого репозитория: https://github.com/anton-petrov/msdtm/tree/master/labs/.

4.2.1 Подготовка к выполнению ЛР№3

В качестве стартового проекта необходимо взять проект ЛР№2, скопировав его и удалив старый локальный репозиторий (каталог «**.git**»). Рекомендуется после этого сразу создать новый локальный репозиторий (команда «**git init**») и сохранить его в новом проекте удалённого репозитория (см. Приложение Б).

Далее рекомендуется взять уровень инфраструктуры (файлы «**l4_InfrastructureLayer.h**» и «**l4_InfrastructureLayer.cpp**») из шаблонного примера при необходимости добавив туда те изменения, которые были сделаны в ЛР№2 (если таковые были).

В эти файлы была добавлена реализация классов «**Exception**» и «**LogSession**», которые пригодятся для дальнейшей работы.

Также необходимо скопировать из шаблонного примера каталог «**monitoring**» и файлы «**.gitignore**» и «**.gitlab-ci.yml**», при необходимости восстановив в них изменения, необходимые для работы проекта.

Рекомендуется после этого проверить работоспособность проекта и сохранить его в удалённом репозитории. Старые тесты должны выполняться.

4.2.2 Изменение кода программы

Во-первых, рекомендуется изменить файл «**l2_ApplicationLayer.h**», добавив в метод «**Application::performCommand**» параметр с номером сессии (см. Рисунок 3). Номер сессии генерируется как можно ранее в цепочке вызовов микросервисов, чтобы в дальнейшем можно было отследить как выполнялся запрос, что особенно актуально при работе большого количества микросервисов (такое может произойти при расширении функций нашего приложения в будущем, но подготовить структуру кода для этого желательно на раннем этапе).

```
 1  /*
 7  */
 8
 9  #ifndef HW_L2_APPLICATION_LAYER_H
10  #define HW_L2_APPLICATION_LAYER_H
11
12  #include <string>
13
14  #include "hw/l3_DomainLayer.h"
15
16  class IOutput
17  {
18  public:
19      virtual ~IOutput() = default;
20
21      virtual void Output(std::string s) const = 0;
22  };
23
24  class Application
25  {
26  public:
27      Application() = delete;
28      Application(const Application &) = delete;
29
30      Application & operator=(const Application &) = delete;
31
32      Application(const IOutput & out)
33          : _out(out)
34      {}
35
36      bool performCommand(int session_id, const std::vector<std::string> & args);
37
38  protected:
39      void error(LogSession &log, std::string text);
40
41  private:
42      const IOutput & _out;
43      ItemCollector   _col;
44  };
45
46  #endif // HW_L2_APPLICATION_LAYER_H
47
```

Рисунок 3 – Добавляем номер сессии в **Application::performCommand**

Затем нужно сформировать этот номер и передать в исправленный метод «**Application::performCommand**». Для этого нужно исправить код слоя пользовательского интерфейса (см. Рисунок 4).

```
26 int main(int , char **)
27 {
28     TerminalOutput  out;
29     Application     app(out);
30     int             session_id = LogSession::generateSessionID();
31
32     for (string line; getline(cin, line); )
33     {
34         if (line.empty())
35             break;
36
37         istringstream  iss(line);
38         vector<string> args;
39
40         for(string str; iss.good();)
41         {
42             iss >> str;
43             args.emplace_back(str);
44         }
45
46         if (!app.performCommand(session_id, args))
47             return 1;
48     }
49
50     cout << "Выполнение завершено успешно" << endl;
51     return 0;
52 }
```

Рисунок 4 – Формирование номера сессии в UI и передача в AL

Далее вносим изменения в сам метод «**Application::performCommand**», в котором, помимо добавления параметра с номером сессии добавляем обработку исключений (см. Рисунок 5 и Рисунок 6).

```
 9  bool Application::performCommand(int session_id, const vector<string> & args)
10  {
11      if (args.empty())
12          return false;
13
14      LogSession log(session_id, args[0]);
15
16      try
17      {
18          if (args[0] == "l" || args[0] == "load")
19          {
20              string filename = (args.size() == 1) ? "hw.data" : args[1];
21
22              if (!_col.loadCollection(filename))
23              {
24                  error(log, "Ошибка при загрузке файла '" + filename + "'");
25                  return false;
26              }
27
28              return true;
29          }
30
31          if (args[0] == "s" || args[0] == "save")
32          {
33              string filename = (args.size() == 1) ? "hw.data" : args[1];
34
35              if (!_col.saveCollection(filename))
36              {
37                  error(log, "Ошибка при сохранении файла '" + filename + "'");
38                  return false;
39              }
40
41              return true;
42          }
```

Рисунок 5 – Изменения в **performCommand** (начало)

```
199                 if (!_col.isRemoved(i))
200                 {
201                     uint64_t debt_sum = 0;
202 
203                     for(const auto & [date,debt] : person.getDebtStory())
204                         if (!debt.didRepair())
205                             debt_sum += debt.getValue();
206 
207                     _out.Output("[" + to_string(i) + "] "
208                             + person.getFirstName() + " "
209                             + person.getLastName() + " "
210                             + to_string(debt_sum) + " "
211                             + person.getEmail());
212 
213                     count ++;
214                 }
215             }
216 
217             _out.Output("Количество не оповещённых должников на " + today.getDateString()
218                     + " : " + to_string(count));
219             return true;
220         }
221     }
222     catch (const Exception & e)
223     {
224         error(log, "Произошло внутреннее (" + e.where() +
225                    ") исключение при обработке оператора " + args[0] +
226                    ": " + e.what());
227         return false;
228     }
229     catch (const exception & e)
230     {
231         error(log, "Произошло исключение при обработке оператора " + args[0] +
232                    ": " + e.what());
233         return false;
234     }
235     catch (...)
236     {
237         error(log, "Произошло исключение при обработке оператора " + args[0]);
238         return false;
239     }
240 
241     error(log, "Недопустимая команда '" + args[0] + "'");
242     return false;
243 }
```

Рисунок 6 – Изменения в **performCommand** (окончание)

Также удобно создать единый метод вывода сообщений об ошибках (см. Рисунок 7).

```
245 void Application::error(LogSession &log, string text)
246 {
247     log.error(text);
248     _out.Output(text);
249 }
250 
```

Рисунок 7 – Application::error

И наконец, нужно заменить все вызовы **assert** на генерацию исключений – наш микросервис не должен падать из-за нарушений инвариантов классов (см. Рисунок 8).

```
63 Debt::Debt(uint64_t v, Date d, string bn, bool r)
64     : _value(v), _date(d), _bank_name(bn), _did_repaid(r)
65 {
66     if (!invariant())
67         throw Exception("Debt::Debt","invariant");
68 }
69
70
71 Person::Person(const std::string & first_name, const std::string & last_name, string email)
72     : _first_name(first_name)
73     , _last_name(last_name)
74     , _email(email)
75 {
76     if (!invariant())
77         throw Exception("Person::Person 1","invariant");
78 }
79
80 Person::Person(const string &first_name, const string &last_name, string email,
81                std::map<Date, Debt> debt_story,
82                set<Date> sending_alerts)
83     : _first_name(first_name)
84     , _last_name(last_name)
85     , _email(email)
86     , _debt_story(debt_story)
87     , _sending_alerts(sending_alerts)
88 {
89     if (!invariant())
90         throw Exception("Person::Person 2","invariant");
91 }
```

Рисунок 8 – Заменяем assert на throw Exception

Помимо этого, нужно создать три теста с проверкой обработки ошибочных ситуаций: «**10.test**», «**11.test**» и «**12.test**».

Перед запуском тестов в рабочем каталоге необходимо создать папку «**tmp**», куда выкладывается журнал работы программы (см. Рисунок 9).

```
54 2020-05-11 12:57:36 34608 END 10
55 2020-05-11 12:57:36 34608 START as
56 2020-05-11 12:57:36 34608 END 10
57 2020-05-11 12:57:36 34608 START s
58 2020-05-11 12:57:36 34608 END 129
59 2020-05-11 12:57:36 34608 START c
60 2020-05-11 12:57:36 34608 END 12
61 2020-05-11 12:57:36 34608 START l
62 2020-05-11 12:57:36 34608 END 40
63 2020-05-11 12:57:36 34608 START v
64 2020-05-11 12:57:36 34608 END 187
65 2020-05-11 12:57:37 24461 START l
66 2020-05-11 12:57:37 24461 END 385
67 2020-05-11 12:57:37 24461 START vd
68 2020-05-11 12:57:37 24461 END 154
69 2020-05-11 12:57:37 24461 START vd
70 2020-05-11 12:57:37 24461 END 83
71 2020-05-11 12:57:40 5282 START a
72 2020-05-11 12:57:40 5282 ERROR Некорректное количество аргументов команды add
73 2020-05-11 12:57:40 5282 END 97
74 2020-05-11 12:57:40 52967 START l
75 2020-05-11 12:57:40 52967 ERROR Ошибка при загрузке файла 'Иван'
76 2020-05-11 12:57:40 52967 END 178
77 2020-05-11 12:57:42 83585 START l
78 2020-05-11 12:57:42 83585 END 107
79 2020-05-11 12:57:42 83585 START vd
80 2020-05-11 12:57:42 83585 ERROR Произошло внутреннее (Date::Date(string str)) исключение г
81 2020-05-11 12:57:42 83585 END 61
82 2020-05-11 12:57:42 8765 START a
```

Рисунок 9 – Журнал работы приложения

Рекомендуется после этого проверить работоспособность проекта и сохранить его в удалённом репозитории. Старые тесты должны выполняться.

4.2.3 Запуск тестов и мониторинг

Перед запуском тестов с мониторингом необходимо выполнить следующие предварительные операции:

– скомпилировать программу формирования данных для отображения в виде диаграммы;

– установить утилиту отображения результатов **gnuplot**.

Для компиляции программы формирования данных нужно в каталоге проекта выполнить следующую команду:

```
g++ -O2 -o bin/databuilder monitoring/databuilder.cpp
```

Для установки утилиты отображения результатов нужно выполнить следующие команды:

```
sudo apt-get update
sudo apt-get install gnuplot
```

Если предыдущий этап был выполнен успешно, можно выполнить тест и мониторинг, запустив команду:

```
monitoring/latency-test
```

В каталоге «**tmp**» должен появиться файл результатов мониторинга «**plot.png**» (см. Рисунок 10).

Рисунок 10 – Результаты мониторинга программы

Для лучшего понимания как работает приложение рекомендуется изучить файлы каталога «**monitoring**», также файл «**.gitlab-ci.yml**».

4.3 Варианты заданий

Для лабораторной работы №3 не предусмотрено вариантов заданий. Базой для выполнения данной работы является предыдущая лабораторная работа №2.

4.4 Требования к отчёту

Отчет должен содержать:

– номер и текст задачи/карточки ЛР№1;

– номер и текст задания для ЛР№2;

– диаграмма мониторинга, получаемая автоматически во время непрерывной интеграции и условного развертывания приложения ЛР№3 в среде ранера;

– выводы.

Заготовку проекта можно взять из репозитория по адресу https://github.com/anton-petrov/msdtm/tree/master/labs.

Лабораторная работа №3 должна быть сохранена в репозитории с использованием CI. Отчёт должен быть выполнен в файле описания проекта «README.md». Предоставлять бумажный отчет не нужно.

4.5 Контрольные вопросы

После выполнения данной контрольной работы нужно уметь ответить на следующие контрольные вопросы:

1) Дайте определение DevOps.
2) Назовите цели DevOps.
3) Из каких составных частей состоит DevOps?
4) Зачем нужно журналирование в рамках DevOps?
5) Зачем нужно генерировать и использовать номер сессии?
6) Почему для работы микросервисов не желательно использовать функцию assert?
7) Зачем нужен мониторинг в рамках DevOps?
8) Приведите пример приложений, для которых целесообразна методология DevOps.

5 Литература

1 Иванова Г.С. Программирование: Учеб. для вузов. – М.: «Кнорус», 2014.

2 Иванова Г.С. Ничушкина Т.Н., Объектно-ориентированное программирование. Учеб. для вузов. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2014.

3 Иванова Г.С. Технология программирования: Учеб. для вузов. – М.: «Кнорус», 2013.

4 Буч Г. Объектно-ориентированное проектирование с примерами применения. – М.: Конкорд, 1992. – 519 с.

5 Гамма Э., Хелм Р., Джонсон Р. [и др.]. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2001. – 366 с.

6 Мартин, Роберт С. Гибкая разработка программ на Java и C++: принципы паттерны и методики. – Спб. : ООО «Диалектика», 2019. – 704 с.

7 Холл, Гэри Маклин. Адаптивный код: гибкое кодирование с помощью паттернов проектирования и принципов SOLID, 2-е изд. – СПб.: ООО «Диалектика», 2018. – 448 с.

8 Эванс, Эрик. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем. – Спб. ООО «Диалектика», 2019. – 448 с.

9 GitHub.com. – URL: https://github.com

10 ISO/IEC 14882:2017 Programming languages — C++. URL: https://www.iso.org/standard/68564.html

11 Ньюмен С. Создание микросервисов. — СПб.: Питер, 2016. — 304 с.: ил. — (Серия «Бестселлеры O’Reilly»).

12 Парминдер Сингх Кочер. Микросервисы и контейнеры Docker. — ДМК Пресс, 2019

13 Мартин Клеппман. Высоко-нагруженные приложения. Программирование, масштабирование, поддержка. — Спб.: Питер, 2020. — 640 с.: ил. — (Серия «Бестселлеры O`Reilly»)

14 Microservice Architecture. URL: https://microservices.io

15 Pattern: Microservice Architecture. URL: https://microservices.io/patterns/microservices.html

16 Джез Хамбл, Дейвид Фарли. Непрерывное развёртывание ПО: автоматизация процессов сборки, тестирования и внедрения новых версий программ. — М.: ООО «И.Д. Вильямс», 2017. — 432 с.

17 Джордж Спаффорд, Джин Ким, Кевин Бер. Проект «Феникс». Роман о том, как DevOps меняет бизнес к лучшему. Серия «Роман в стиле бизнес». – Эксмо; Москва; 2015

Приложение А. Соглашение о стиле кодирования

Таблица 6 – Перечень основных правил соглашения о стиле кодирования

№№	Правило	Пояснения
1. Запрещающие соглашения		
1.1.	Следует избегать использования глобальных переменных	
1.2.	Следует избегать использования макросов	
1.3.	Следует избегать использования оператора **`goto`**	
1.4.	Следует избегать использования указателей C++	Используйте интеллектуальные указатели: unique_ptr, shared_ptr, weak_ptr
1.5.	Следует избегать использования массивов C++	Используйте параметризованные контейнеры стандартной библиотеки C++: arrey, vector, deque и другие
2. Соглашения об именовании		
2.1.	Имена, представляющие типы, должны быть написаны в смешанном регистре, начиная с верхнего	Пример: `Line, SavingsAccount`

2.2.	Имена переменных должны быть записаны в нижнем регистре, слова должны разделяться знаком подчеркивания	Пример:
		`line, savings_account`
		Распространённая в настоящее время практика в сообществе разработчиков C++. Позволяет легко отличать переменные от типов, предотвращает потенциальные коллизии имён, например:
		`Line line;`
		Также, записанные таким образом переменные легко отличить от названия метода (см. далее).

2.3.	Названия методов и функций должны быть глаголами, быть записанными в смешанном регистре и начинаться с нижнего	Пример: `getName(), computeTotalWidth()`

№№	Правило	Пояснения

2.4.	Аббревиатуры и сокращения в именах должны записываться в нижнем регистре	Пример:
		`exportHtmlSource();`
		`// НЕЛЬЗЯ: exportHTMLSource();`
		`openDvdPlayer();`
		`// НЕЛЬЗЯ: openDVDPlayer();`
		Использование верхнего регистра может привести к конфликту имён, описанному выше. Иначе переменные бы имели имена dVD, hTML и т. д., что не является удобочитаемым. Другая проблема уже описана выше; когда имя связано с другим, читаемость снижается; слово, следующее за аббревиатурой, не выделяется так, как следовало бы.

2.5.	Членам класса с модификатором private следует присваивать префикс-подчёркивание	Пример:
		`class SomeClass`
		`{`
		`private:`
		`    int _length;`
		`}`
		Не считая имени и типа, область видимости — наиболее важное свойство переменной. Явное указание модификатора доступа в виде подчёркивания избавляет от путаницы между членами класса и локальными переменными. Это важно, поскольку переменные класса имеют большее значение, нежели переменные в методах, и к ним следует относиться более осторожно.
		Дополнительным эффектом от префикса-подчёркивания является разрешение проблемы именования в методах, устанавливающих значения, а также в конструкторах:
		`void setDepth (int depth)`
		`{`
		`    _depth = depth;`
		`}`

2.6.	Все имена следует записывать по-английски	Пример:
		`file_name;`
		`// НЕ РЕКОМЕНДУЕТСЯ: imyaFayla`

№№	Правило	Пояснения
2.7.	Переменные, имеющие большую область видимости, следует называть длинными именами, имеющие небольшую область видимости — короткими	Имена временных переменных, использующихся для хранения временных значений или индексов, лучше всего делать короткими. Программист, читающий такие переменные, должен иметь возможность предположить, что их значения не используются за пределами нескольких строк кода. Обычно это переменные i, j, k, l, m, n (для целых), а также c и d (для символов).

2.8.	Имена объектов не указываются явно, следует избегать указания названий объектов в именах методов	Пример:
		`line.getLength();`
		`// НЕ РЕКОМЕНДУЕТСЯ:`
		`line.getLineLength();`
		Второй вариант смотрится вполне естественно в объявлении класса, но совершенно избыточен при использовании, как это и показано в примере.

3. Особые правила наименования		

3.1.	Слова get/set должны быть использованы везде, где осуществляется прямой доступ к атрибуту	Пример:
		`employee.getName();`
		`employee.setName(name);`
		
		`matrix.getElement(2, 4);`
		`matrix.setElement(2, 4, value);`

3.2.	Множественное число следует использовать для представления наборов (коллекций) объектов	Пример:
		`vector<Point>  points;`
		`int  values[];`
		Улучшает читаемость, поскольку имя даёт пользователю прямую подсказку о типе переменной и операциях, которые могут быть применены к этим элементам.

3.3.	Префикс n следует использовать для представления числа объектов	Пример: `n_points, n_lines` Обозначение взято из математики, где оно является установившимся соглашением для обозначения числа объектов.
3.4.	Суффикс No следует использовать для обозначения номера сущности	Пример: `table_no, employee_no` Обозначение взято из математики, где оно является установившимся соглашением для обозначения номера сущности.

№№	Правило	Пояснения

3.5.	Переменным-итераторам следует давать имена i, j, k и т. д. Или it, чтобы не путать его с целым типом.	Пример:
		`for(int i = 0; i < n_tables); i++)`
		`{`
		`    ...`
		`}`
		Обозначение взято из математики, где оно является установившимся соглашением для обозначения итераторов.
		`for(auto it = list.begin(); it != list.end(); ++it)`
		`{`
		`    Element element = *it;`
		`    ...`
		`}`

3.6.	Префикс is следует использовать только для булевых (логических) переменных и методов	Пример:
		`s_set, is_visible, is_finished, is_found, is_open`
		Использование этого префикса избавляет от таких имён, как status или flag. is_status или is_flag просто не подходят, и программист вынужден выбирать более осмысленные имена.
		В некоторых ситуациях префикс is лучше заменить на другой: has, can или should:
		`bool has_license();`
		`bool can_evaluate();`
		`bool should_sort();`

4. Файлы исходных кодов		
4.1.	Заголовочным файлам C++ следует давать расширение .h (предпочтительно) либо .hpp. Файлы исходных кодов могут иметь расширения .cpp. Имена файлов записываются в нижнем регистре	Пример: `myclass.cpp, myclass.h` Эти расширения, одобряемые стандартом C++.

№№	Правило	Пояснения
4.2.	Класс следует объявлять в заголовочном файле и определять (реализовывать) в файле исходного кода, имена файлов совпадают с именем класса (но пишутся в нижнем регистре)	Пример: `myclass.cpp, myclass.h` Облегчает поиск связанных с классом файлов. Очевидное исключение — шаблонные классы, которые должны быть объявлены и определены в заголовочном файле.
4.3.	Нельзя использовать специальные символы (например, TAB) и разрывы страниц	Такие символы вызывают ряд проблем, связанных с редакторами, эмуляторами терминалов и отладчиками, используемыми в программах для совместной разработки и кроссплатформенных средах.

4.4.	Заголовочные файлы должны содержать защиту от вложенного включения	Пример:
		`#ifndef _COM_COMPANY_MODULE_CLASSNAME_H_`
		`#define _COM_COMPANY_MODULE_CLASSNAME_H_`
		`...`
		`#endif // _COM_COMPANY_MODULE_CLASSNAME_H_`
		Конструкция позволяет избегать ошибок компиляции. Это соглашение позволяет увидеть положение файла в структуре проекта и предотвращает конфликты имён.

4.5.	Директивы включения следует сортировать (по месту в иерархии системы, ниже уровень — выше позиция) и группировать. Оставляйте пустую строку между группами	Прррмер:
		`#include <fstream>`
		`#include <iomanip>`
		
		`#include <qt/qbutton.h>`
		`#include <qt/qtextfield.h>`
		
		`#include "com/company/ui/PropertiesDialog.h"`
		`#include "com/company/ui/MainWindow.h"`
		Пути включения не должны быть абсолютными. Вместо этого следует использовать директивы компилятора.

4.6.	Директивы включения должны располагаться только в начале файла	Общая практика. Избегайте нежелательных побочных эффектов, которые может вызвать «скрытое» включение где-то в середине файла исходного кода.
5. Разное		
5.1.	Следует избегать «магических» чисел в коде. Числа, отличные от 0 или 1, следует объявлять, как именованные константы	Если число само по себе не имеет очевидного значения, читаемость улучшается путём введения именованной константы. Другой подход — создание метода, с помощью которого можно было бы осуществлять доступ к константе.

№№	Правило	Пояснения
5.2.	Следует использовать null_ptr вместо «NULL»	NULL является частью стандартной библиотеки C и устарело в C++
6. Оформление и комментарии		

6.1.	Основной отступ следует делать в четыре пробела	Пример:
		`for(i = 0; i < n_elements; i++)`
		`    a[i] = 0;`
		Отступ в один или два пробела достаточно мал, чтобы отражать логическую структуру кода. Отступ более 4 пробелов делает глубоко вложенный код нечитаемым и увеличивает вероятность того, что строки придётся разбивать. Широко распространены варианты в 2, 3 или 4 пробела; причём 2 и 4 — более широко.

6.2.	Блоки кода следует оформлять так, как показано в примере 1 (рекомендуется), но ни в коем случае не так, как показано в примере 2. Оформление функций и классов должно следовать примеру 1	Пример 1:
		`while(!done)`
		`{`
		`    doSomething();`
		`    done = moreToDo();`
		`}`
		
		Пример 2:
		`while(!done)`
		`  {`
		`    doSomething();`
		`    done = moreToDo();`
		`  }`
		Пример 2 использует лишние отступы, что мешает ясному отображению логической структуры кода.

Приложение Б. Настройка работы с вузовским репозиторием кода

В таблице 7 приводится последовательность действий для установки и настройки системы Git, а также подключения к GitLab (GitLab) и настройки работы с проектами в этой системе.

Таблица 7 – Последовательность действий для настройки работы с вузовским репозиторием кода

№№	Действие	Пояснение
1.	Установка Git на рабочий компьютер	Описание установки Git можно почитать на следующей странице: https://git-scm.com/book/ru/v1/Введение-Установка-Git. После завершения установки нужно проверить успешность ее выполнения, введя в терминале (или в Git Bash, если вы ставили в ОС Windows) следующую команду: `git --version` В результате ее выполнения должна вывестись версия установленной системы Git. Текущая версия: 2.20.

2.	Конфигурация Git	После установки Git необходимо указать реквизиты пользователя. Для этого нужно ввести следующие команды:
		`git config --global user.name "<ФИО>"`
		`git config --global user.email "<почта>"`
		, где <ФИО> – ваша фамилия и инициалы,
		<почта> – адрес вашей электронной почты.

3.	Получение открытого ключа SSH	Работа по протоколу SSH является предпочтительной. Для получения открытого ключа SSH нужно сначала проверить его наличие, возможно он у вас установлен. Для этого в ОС GNU/Linux нужно найти скрытый каталог `.ssh` и убедиться в наличии в нем файла `id_rsa.pub`. Открытый ключ SSH является содержимым данного файла. Если указанного каталога или файла нет, то нужно выполнить генерацию ключей SSH. Как это сделать можно почитать на следующей странице: Git-на-сервере-Генерация-открытого-SSH-ключа.

№№	Действие	Пояснение
4.	Регистрация на вузовском репозитории исходного кода и регистрация открытого SSH-ключа	После регистрации на репозитории исходного кода (https://github.com) нужно зайти в личный кабинет (пункт «Настройки»), выбрать в левой панели пункт «SSH-ключи» и добавить открытый ключ SSH.

5.	Копирование или клонирование заготовки для выполнения лабораторной работы №1 на локальный компьютер	Заготовку лабораторной работы №1 можно просто скопировать из репозитория или склонировать.
		Перед выполнением клонирования рекомендуется перейти в каталог, в котором планируется разместить каталог проекта. Переход в этот каталог осуществляется с помощью следующей команды:
		`cd <путь>`
		, где <путь> – имя каталога, в котором планируется разместить каталог проекта.
		Для выполнения лабораторной работы №1 нужно выполнить клонирование заготовки этой лабораторной работы из репозитория исходного кода:
		`git clone https://github.com/anton-petrov/msdtm.git`
		При этом в текущем каталоге образуется подкаталог, в который скопируется проект из репозитория.
		Лабораторная работа №2 выполняется на основе выполненной лабораторной работы №1.

6.	Получение доступа к группе с настроенным механизмом непрерывной интеграции	После регистрации в вузовском репозитории исходного кода нужно выслать преподавателю придуманный вами никнейм, чтобы предоставить вам доступ к группе домашних заданий и лабораторных работ. Пример наименования группы: «PI-2041».
7.	Создание проекта в GitLab	После получения доступа к группе домашних заданий и лабораторных работ нужно зайти в нее и создать новый проект, наименование которого должно быть следующего формата: XXX-IOFamilia, Где XXX – номер группы, IO – инициалы, Familia – фамилия. Инициалы, фамилия и все остальные буквы должны быть из английского алфавита (фамилия – созвучная вашей на английском языке).

№№	Действие	Пояснение
8.	Настройка удаленного репозитория для вашего проекта	Для подключения удаленного репозитория нужно в терминале перейти в каталог с проектом (с помощью команды `cd <путь>`) и там ввести следующую команду: `git remote add origin <адрес проекта>` , где `<адрес проекта>` – git-адрес вашего проекта, который используется для клонирования (в пустом проекте на GitLab можно посмотреть в подсказках)

9.	Сохранение изменений в удаленном репозитории	После того, как будут произведены какие-либо изменения в исходном коде, нужно сохранить их в локальном, а затем в удаленном репозитории. Для итого нужно, находясь в терминале в каталоге проекта, последовательно ввести следующие команды:
		– Добавление изменений в индекс репозитория:
		`git add .`
		– Добавление изменений в репозиторий:
		`git commit -m "XXX"`
		, где XXX – комментарий к изменениям
		– Сохранение изменений в удаленный репозиторий:
		`git push -u origin master`
		– при первом добавлении и:
		`git push`
		– при последующих.
		Старайтесь чаще делать сохранение изменений.

10.	Работа непрерывной интеграции (CI)	При сохранении изменений в удаленный репозиторий для проектов группы домашних заданий и лабораторных работ выполняется непрерывная интеграция, которая состоит из следующих этапов (см. файл `.gitlab-ci.yml`): Компиляция. Компиляция проводится компилятором GNU C++ из пакета GCC (GNU Compiler Collection). Прогон автотестов. Генерация описания классов проекта и выкладывание его в page-область проекта. Посмотреть эту область можно через левую панель, пункт «Настройка» / «Страницы», где указан путь к page-области. Если хотя бы один из этапов выполнен с ошибками, считается, что сборка проекта не удалась.

Перечень условных обозначений, сокращений и терминов

ITIL	IT Infrastructure Library – библиотека инфраструктуры информационных технологий
RAII	«Resource Acquisition Is Initialization», бук. «получение ресурса есть инициализация» — техника использования механизма контроля зоны видимости объекта для управления ресурсами, которые он содержит
SOLID	Мнемонический акроним для пяти основных принципов объектно-ориентированного проектирования
ЛР	Лабораторная работа
ООП	Объектно-ориентированное проектирование
ОПКС	Собственные общепрофессиональные компетенции
ПО	Программное обеспечение
ЯП	Язык программирования