

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ**
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Кемеровский государственный университет»
Институт цифры
Кафедра ЮНЕСКО по информационным технологиям

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ
ПО ДИСЦИПЛИНЕ
“Программная инженерия”

«Лабораторная работа №3 Лабораторные работы МиСПИ»

студента 3 курса
Копытова Егора Андреевича ПИ-203
Направление 09.03.03 – Прикладная информатика в экономике

Руководитель профессор:
Ю.А.Степанов

Работа защищена:
“ ” _____ 2023г.

Кемерово 2023 г.

Лабораторная работа №1

1. Документ Software Requirements Specification, содержащий список требований к сайту.
2. UseCase-диаграммы прецедентов использования, реализующих функциональные требования.
3. Выводы по работе.

1.UseCase – диаграмма прецедентов.

1.2 Operating environment

- 1.2.1 Сайт поддерживается браузерами Firefox, Opera, Chrome, Яндекс и другие виды.
- 1.2.2 Front – реализован при помощи HTML, CSS, JS
- 1.2.3 Back – реализован на PHP

1.3 User documentation

Документация не требуется

Для создания диаграмм прецедентов был выбран сайт по чтению манги. (Remanga.org: <https://remanga.org/>). Рассмотрим диаграмму прецедентов на рисунке 1.

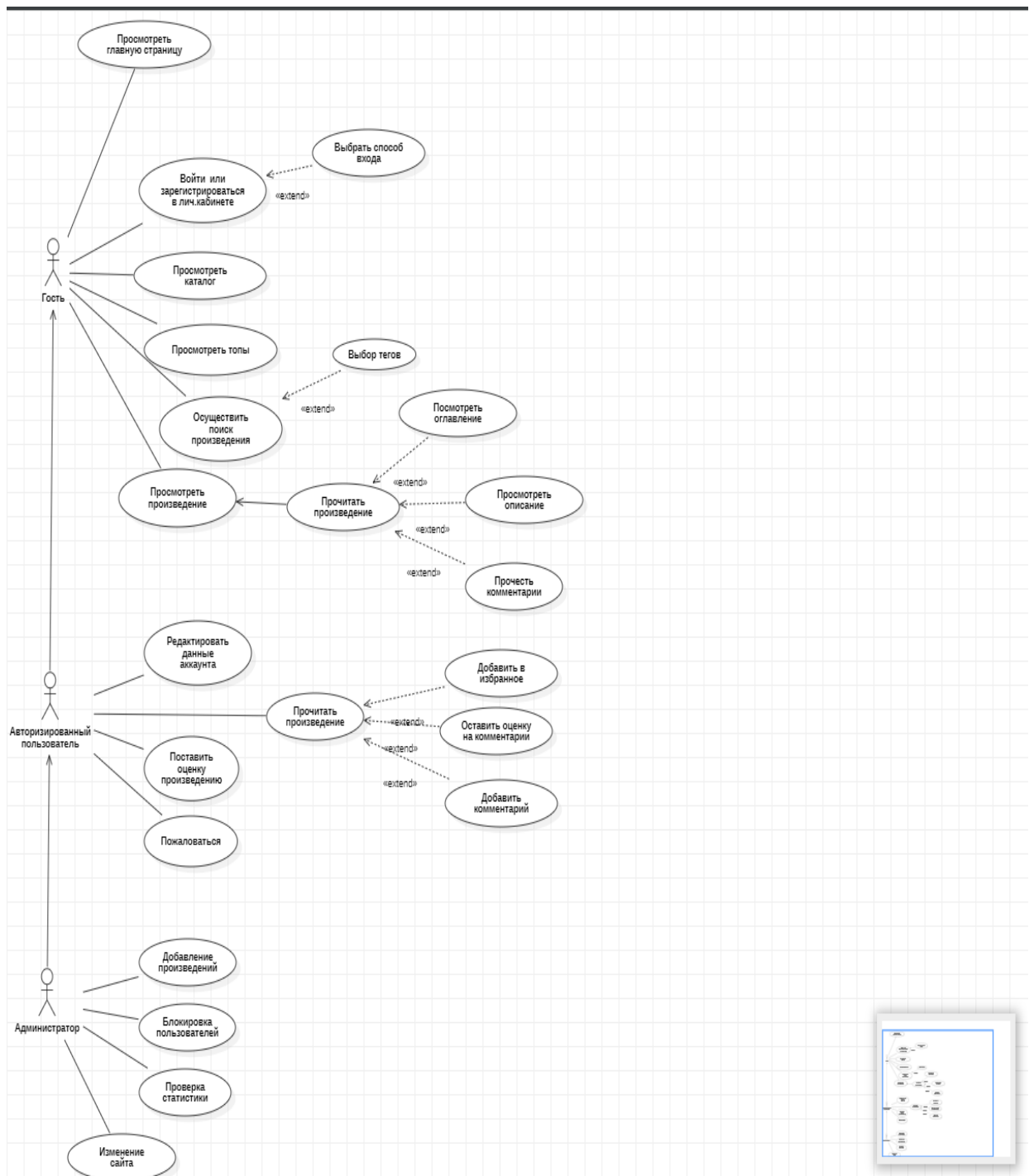


Рисунок 1 UseCase диаграмма сайта

3. Specific requirements

3.1 Functional requirements

Требования владельца сайта:

FR1. Система должна предоставлять возможность добавления новых произведений.

FR2. Система должна предоставлять возможность авторизации в роли администратора.

FR3. Система должна предоставлять функционал просмотра и учета статистики.

FR4. Система должна предоставлять возможности блокировки пользователей.

FR5. Система должна предоставлять возможность изменения сайта.

Требования пользователей сайта:

FR6. Система должна предоставлять функционал регистрации на сайте.

FR7. Система должна предоставлять функционал авторизации пользователя.

FR8. Система должна предоставлять функционал редактирования данных аккаунта пользователя.

SEC1. Система должна предоставлять возможность при утрате пароля его восстановить, а также возможность выбора входа.

FR9. Система должна предоставлять возможность поиска произведения по названию, автору, тегу.

FR10. Система должна предоставлять возможность просмотра Топов произведений по рейтингу сайта и перехода на последующий сайт.

FR11. Система должна предоставлять возможность просмотреть список недавно добавленных произведений.

FR12. Система должна предоставлять возможность прочитать выбранное произведение на русском языке.

FR13. Система должна предоставлять возможность просмотреть оглавление произведения и перейти на соответствующую главу.

FR14. Система должна предоставлять возможность регулировать тематику сайта.

FR15. Система должна предоставлять возможность отметить произведение как прочитанное.

FR16. Система должна предоставлять возможность запомнить страницу и добавить ее в закладки.

FR17. Система должна предоставлять возможность поставить оценку произведению.

FR18. Система должна предоставлять возможность перехода на страницу выбранного произведения.

FR19. Система должна предоставлять возможность поделиться ссылкой произведения на сайте.

FR20. Система должна предоставлять возможность просмотра последних отзывов и перехода на соответствующие произведение.

FR21. Система должна предоставлять возможность оставить отзыв о произведение.

1.1 Non-functional requirements

NFUR – Non-functional usability requirements

NFRR – Non-functional reliability requirements

NFPR – Non-functional performance requirements

NFSR – Non-functional supportability requirements

NFRR1. Сайт должен корректно работать в Firefox, Opera, Chrome, Яндекс и других видах.

NFPR1. Сайт должен корректно работать при пиковых нагрузках (до 40 запросов/сек)

NFSR1. Сайт должен автоматически возобновлять свою работу при непредвиденном отключении в течение часа

NFUR1. Время отклика сайта должно не превышать 3 секунд

NFUR2. URL-адрес сайта – <https://remanga.org/>

NFUR3. Доступный просмотр списка выбранных произведений

NFUR4. В навигационной панели есть ссылки на все категории страниц сайта.

NFUR5. В навигационной панели есть разные режимы подсветки сайта и настройки

ID	Требование	Полезность	Трудоемкость, человеко-час	Стабильность
Functional requirements (владелец сайта):				
FR1	Система должна предоставлять возможность добавления новых произведений.	Важная	5	Средняя
FR2	Система должна предоставлять возможность	Полезная	6	Низкая

	авторизации в роли администратора.			
FR3	Система должна предоставлять функционал просмотра и учета статистики.	Критическая	7	Высокая
FR4	Система должна предоставлять возможности блокировки пользователей.	Важная	4	Высокая
FR5	Система должна предоставлять возможность изменения сайта.	Критическая	6	Высокая
Functional requirements (пользователей сайта):				
FR6	Система должна предоставлять функционал регистрации на сайте.	Полезная	2	Низкая
FR7	Система должна предоставлять функционал авторизации пользователя.	Важная	4	Средняя

FR8	Система должна предоставлять функционал редактирования данных аккаунта пользователя.	Полезная	2	Средняя
FR9	Система должна предоставлять возможность поиска произведения по названию, автору, тегу.	Критическая	7	Средняя
FR10	Система должна предоставлять возможность просмотра Топов произведений по рейтингу сайта и перехода на последующий сайт.	Полезная	3	Средняя
FR11	Система должна предоставлять возможность просмотреть список недавно добавленных произведений.	Полезная	3	Низкая

FR12	Система должна предоставлять возможность прочитать выбранное произведение на русском языке.	Критическая	6	Высокая
FR13	Система должна предоставлять возможность просмотреть оглавление произведения и перейти на соответствующую главу.	Важная	4	Низкая
FR14	Система должна предоставлять возможность регулировать тематику сайта.	Критическая	12	Высокая
FR15	Система должна предоставлять возможность отметить произведение как прочитанное.	Полезная	2	Низкая
FR16	Система должна предоставлять	Полезная	4	Средняя

	возможность запомнить страницу и добавить ее в закладки.			
FR17	Система должна предоставлять возможность поставить оценку произведению.	Полезная	2	Средняя
FR18	Система должна предоставлять возможность перехода на страницу выбранного произведения.	Полезная	7	Средняя
FR19	Система должна предоставлять возможность поделиться ссылкой произведения на сайте.	Полезная	3	Средняя
FR20	Система должна предоставлять возможность просмотра последних отзывов и перехода на	Критическ ая	7	Низкая

	соответствующие произведение.				
FR21	Система должна предоставлять возможность оставить отзыв о произведение.	Полезная	3	Высокая	
Non-functional requirements:					
NFRR1	Сайт должен корректно работать при пиковых нагрузках (до 40 запросов/сек)	Критическ ая	20	Низкая	
NFPR1	Сайт должен корректно работать в Firefox, Opera, Chrome, Яндекс и других видах.	Критическ ая	20	Средняя	
NFSR1	Сайт должен автоматически возобновлять свою работу при непредвиденном отключении в течение часа	Критическ ая	15	Низкая	

NFUR1	Время отклика сайта должно не превышать 3 секунд	Важная	20	Средняя
NFUR2	URL-адрес сайта – https://remanga.org/	Важная	3	Низкая
NFUR3	Доступный просмотр списка выбранных произведений	Важная	3	Высокая
NFUR4	В навигационной панели есть ссылки на все категории страниц сайта.	Важная	2	Средняя
NFUR5	В навигационной панели есть разные режимы подсветки сайта и настройки	Важная	2	Средняя

РИСКИ

Риск	Вероятность	Последствия	Список требований, подверженных влиянию
Недостаток квалификации сотрудников	Низкая	-Увеличение количества багов	FR 1-21, NFRR1, NFPR1, NFSR1, NFUR 1-5

		<p>-Увеличение времени реализации проекта:</p> <p>-Увеличение затрат на исправление допущенных ошибок</p> <p>-Необходимость тратить время на повышение квалификации сотрудников</p>	
Недостаток финансирования проекта	Средний	<p>- Распад команды</p> <p>-Недостаток оборудования</p> <p>-Недостаток функционала</p> <p>-Недостатки при создании и обслуживании сайта</p>	NFRR1, NFPR1, NFSR1, NFUR1
Недостаток времени на реализацию проекта	Низкая	<p>-Незаконченность проекта</p> <p>- Возникновение ошибок</p>	NFUR 2-4,

		- Потеря потока клиентов	
Смена внешних API	Средняя	<ul style="list-style-type: none"> - Трудны поиск и потеря прошлого места в списки поисков - Потеря связей между различными модулями сайта 	FR5, FR19
Смена концепции предоставления услуг	Низкая	<ul style="list-style-type: none"> - Изменение Всего функционала сайта 	Все
Нехватка клиентов для тестирования	Низкая	<ul style="list-style-type: none"> -Увеличение времени реализации проекта -Падение производительности - Увеличение риска возникновения последующих ошибок 	NFPR1, NFSR1, NFUR 1
Потеря смысловой взаимосвязи интерфейса	Средняя	<ul style="list-style-type: none"> - Дезориентация пользователя вовремя 	FR3

		<p>использования сайта</p> <p>- Увеличение срока выполнения поиска произведения</p> <p>- Отток клиентов</p>	
--	--	---	--

1.2 Use-cases (Прецеденты)

Имя прецедента	Просмотр главной страницы
ID	1
Краткое описание	Просмотр главной страницы сайта с различными представленными рекомендациями
Актёры, вовлеченные в прецедент	Пользователь, Гость
Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Пользователь выбирает значок главного сайта 2. Система переводит пользователя или гостя на главную страницу сайта
Постусловия	Пользователь или гость видит представленные произведения или предложения
Альтернативные потоки	—

Имя прецедента	Войти или зарегистрироваться в личный кабинет
ID	2
Краткое описание	Регистрация нового пользователя или вход уже заведенного пользователя в базу данных, по номеру телефона
Актёры, вовлеченные в прецедент	Пользователь
Основной поток	<p>3. Прецедент начинается, когда Пользователь выбирает опцию «Войти».</p> <p>4. Система запрашивает у Пользователя номер его телефона.</p> <p>5. Пользователь вводит в форму номер своего телефона.</p> <p>6. Если данные Пользователя удовлетворяют, то</p> <p>6.1 Пользователь заходит в систему</p> <p>7. Иначе</p> <p>7.1 Данные Пользователя проходят регистрацию, и он подтверждает свой номер телефона</p> <p>7.2 Данные Пользователя попадают в базу данных</p>

Постусловия	Пользователь получает доступ к своему личному кабинету и изменению различных настроек
Альтернативные потоки	Выбор способа входа

Имя прецедента	Альтернативный поток: выбор способа входа
ID	2.1
Краткое описание	Пользователь может выбирать способ входа по предложенным вариантам
Актёры, вовлеченные в прецедент	Пользователь
Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Пользователь выбирает опцию «Войти». 2. Система запрашивает способ входа в систему 3. Пользователь выбирает один из предложенных вариантов <ol style="list-style-type: none"> 3.1 Пользователь вбивает пароль и заходит 3.2 Пользователь вбивает неверные данные и остается на поле ввода

	<p>3.3 После получения доступа пользователь проходит в личный кабинет</p> <p>4. Пользователь возвращается на главную страницу</p>
Постусловия	Пользователь получает доступ к своему личному кабинету и изменению различных настроек
Альтернативные потоки	-

Имя прецедента	Просмотреть каталог
ID	3
Краткое описание	Выбор каталога на основании предпочтений пользователя
Актёры, вовлеченные в прецедент	Пользователь, гость
Основной поток	<p>1. Прецедент начинается, когда Пользователь выбирает вкладку каталог.</p> <p>2. Пользователь может дальше просмотреть каталог или выбрать понравившейся произведение</p>
Постусловия	Пользователь выбирает произведения находящиеся в каталоги

Альтернативные потоки	-
-----------------------	---

Имя прецедента	Посмотреть топы
ID	4
Краткое описание	Пользователь может просмотреть представленный топовые произведения с оценкой на сайте
Актёры, вовлеченные в прецедент	Пользователь, гость
Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Пользователь выбирает вкладку топов 2. Система показывает имеющиеся топовые произведения, из которых пользователь может выбрать понравившиеся
Постусловия	Пользователь получает возможность влиять на подборку произведений в топе.
Альтернативные потоки	-

Имя прецедента	Осуществить поиск произведения
ID	5

Краткое описание	Поиск произведения по названию, автору тегу и т.п.
Актёры, вовлеченные в прецедент	Пользователь, гость
Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Пользователь вводит необходимые параметры поиска. 2. Если выбранные им параметры есть в системе, то <ol style="list-style-type: none"> 2.1 Пользователь переходит на необходимое произведение 3. Иначе <ol style="list-style-type: none"> 3.1 Пользователь видит, что система не смогла найти необходимое произведение
Постусловия	Пользователь осуществляет поиск произведения по известным ему данным
Альтернативные потоки	Выбор тегов

Имя прецедента	Альтернативный поток: Выбор тегов
ID	5.1
Краткое описание	Выбор критериев поиска по определенным тегам.

Актёры, вовлеченные в прецедент	Пользователь, гость
Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Пользователь выбирает поиск произведения по тегам 2. Если выбранные им параметры есть в системе, то <ol style="list-style-type: none"> 2.1 Пользователь переходит на необходимое произведение 3. Иначе <ol style="list-style-type: none"> 3.1 Пользователь видит, что система не смогла найти необходимое произведение
Постусловия	Пользователь ищет необходимое произведение
Альтернативные потоки	-

Имя прецедента	Просмотреть произведение
ID	6
Краткое описание	Выбрать произведение и просмотреть основные сведения
Актёры, вовлеченные в прецедент	Пользователь, гость

Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Пользователь выбрал произведение 2. Далее пользователи нажимают на данное произведение и получают краткую информацию о нем.
Постусловия	-
Альтернативные потоки	-

Имя прецедента	Прочитать произведение
ID	6.1
Краткое описание	Пользователь выбирает произведение и может прочитать основные сведения о данном произведении.
Актёры, вовлеченные в прецедент	Пользователь, гость
Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Пользователь выбрал нужное произведение 2. Далее пользователь выбирает вкладку прочитать и выбирает главу, с которой хочет начать чтение.

Постусловия	Просмотр оглавлений произведений или работа с самим произведением перед его прочтением
Альтернативные потоки	Просмотр оглавления

Имя прецедента	Альтернативный поток: Просмотр оглавления
ID	6.2
Краткое описание	Пользователь может просматривать оглавление читаемого произведения
Актёры, вовлеченные в прецедент	Пользователь, гость
Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Пользователь читает интересующую его произведение 2. Система позволяет просматривать оглавление, не отрываясь от чтения и не переходя на главный экран.
Постусловия	Чтение и выбор необходимого пользователю произведения
Альтернативные потоки	Просмотреть описание

Имя прецедента	Альтернативный поток: Просмотреть описание
----------------	--

ID	6.3
Краткое описание	Пользователь может просматривать описание читаемого произведения
Актёры, вовлеченные в прецедент	Пользователь, гость
Основной поток	<p>1. Прецедент начинается, когда Пользователь читает интересующую его произведение</p> <p>2. Система позволяет просматривать описание, не отрываясь от чтения и не переходя на главный экран.</p>
Постусловия	Чтение и выбор необходимого пользователю произведения
Альтернативные потоки	Просмотреть комментарии

Имя прецедента	Альтернативный поток: Просмотреть комментарии
ID	6.4
Краткое описание	Пользователь может просматривать комментарии читаемого произведения
Актёры, вовлеченные в прецедент	Пользователь, гость
Основной поток	<p>1. Прецедент начинается, когда Пользователь читает</p>

	<p>интересующую его</p> <p>произведение</p> <p>2. Система позволяет просматривать комментарии, не отрываясь от чтения и не переходя на главный экран.</p>
Постусловия	Чтение и выбор необходимого пользователю произведения
Альтернативные потоки	-

Имя прецедента	Редактировать данные аккаунта
ID	7
Краткое описание	Редактирование профиля Пользователя
Актёры, вовлеченные в прецедент	Пользователь
Основной поток	<p>1. Прецедент начинается, когда Пользователь выбирает опцию «Редактирование профиля» в своем профиле.</p> <p>2. Система запрашивает у Пользователя его имя, дата рождения, пол, E-mail.</p> <p>3. Пользователь вводит в форму имя, дата рождения, пол, E-mail.</p> <p>4. Если данные Пользователя удовлетворяют, то</p>

	<p>4.1 Данные сохраняются</p> <p>5. Иначе</p> <p>5.1 Выдается ошибка о некорректности данных.</p>
Постусловия	Пользователь получает различные возможности, предоставляемые сайтом
Альтернативные потоки	-

Имя прецедента	Поставить оценку произведению
ID	8
Краткое описание	Поставить оценку произведению на выбор
Актёры, вовлеченные в прецедент	Пользователь
Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Пользователь выбирает опцию поставить оценку произведению. 2. Система запрашивает выбор оценки и сохраняет выбор пользователя 3. Система предлагает оставить комментарий или вернуться на главный экран или к произведению

Постусловия	Пользователь получает различные возможности, предоставляемые сайтом
Альтернативные потоки	-

Имя прецедента	Прочитать произведение
ID	9
Краткое описание	Пользователь может прочитать произведение, выбранное ранее
Актёры, вовлеченные в прецедент	Пользователь
Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Пользователь выбирает опцию прочитать произведение. 2. Система переносит пользователя на первую страницу произведения или к оглавлению, в зависимости от выбора пользователя.
Постусловия	Пользователь получает различные возможности, предоставляемые сайтом
Альтернативные потоки	Добавить в избранное

Имя прецедента	Альтернативный поток: Добавить в избранное
----------------	--

ID	9.1
Краткое описание	Пользователь может добавить в избранное читаемое или интересующее произведение
Актёры, вовлеченные в прецедент	Пользователь
Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Пользователь выбирает перейти к произведению. 2. Система на главном экране предоставляет информацию о произведении и дает возможность добавления в избранное
Постусловия	Пользователь получает различные возможности, предоставляемые сайтом
Альтернативные потоки	Оставить оценку на комментарии

Имя прецедента	Альтернативный поток: Оставить оценку на комментарии
ID	9.2
Краткое описание	Пользователь может оставлять оценку на комментарии к произведению или главе
Актёры, вовлеченные в прецедент	Пользователь

Основной поток	<p>1. Прецедент начинается, когда Пользователь выбирает перейти к вкладке комментарии</p> <p>2. Система на главном экране предоставляет различные комментарии других пользователей и самого пользователя (если они имеются)</p> <p>3. Пользователь получает возможность оценки комментариев</p>
Постусловия	Пользователь получает различные возможности, предоставляемые сайтом
Альтернативные потоки	Добавить комментарий

Имя прецедента

Альтернативный поток: Добавить комментарий

ID

9.3

Краткое описание

Пользователь может оставлять комментарии к произведению или главе

Актёры, вовлеченные в прецедент

Пользователь

Основной поток

1. Прецедент начинается, когда Пользователь выбирает

	<p>перейти к вкладке комментарии</p> <p>2. Система на главном экране предоставляет различные комментарии других пользователей и самого пользователя (если они имеются)</p> <p>3. Пользователь может добавлять комментарии или редактировать старые</p>
Постусловия	Пользователь получает различные возможности, предоставляемые сайтом
Альтернативные потоки	-
Имя прецедента	Пожаловаться
ID	10
Краткое описание	Пользователь может пожаловаться на выбранное произведение или комментарий
Актёры, вовлеченные в прецедент	Пользователь
Основной поток	<p>1. Прецедент начинается, когда Пользователь выбирает перейти к вкладке произведения</p>

2. Система на главном экране предоставляет кнопку пожаловаться
3. Пользователь может выбрать определенный критерий или написать свою жалобу

Постусловия

Пользователь получает различные возможности, предоставляемые сайтом

Альтернативные потоки

-

Имя прецедента	Добавление произведений
ID	11
Краткое описание	Администратор сайта может добавлять различные произведения
Актёры, вовлеченные в прецедент	Пользователь, Администратор
Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Администратор собирается добавить новое произведение 2. Администратор добавляет новое произведение в определенный раздел сайта и привязывает различными ссылками

Постусловия	-
Альтернативные потоки	-

Имя прецедента	Блокировка пользователя
ID	12
Краткое описание	Администратор сайта может блокировать различных пользователей сайта
Актёры, вовлеченные в прецедент	Пользователь, Администратор
Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Администратор собирается заблокировать пользователя 2. Администратор блокирует пользователя и выставляет причину блокировки
Постусловия	-
Альтернативные потоки	-

Имя прецедента	Проверка статистики
ID	13
Краткое описание	Администратор сайта может проверять статистику интересующих его параметров сайта
Актёры, вовлеченные в прецедент	Администратор

Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Администратор собирается проверить статистику 2. Администратор проверяет статистику выставляя нужные критерии и получает основную или подробную информацию о выбранных параметрах
Постусловия	-
Альтернативные потоки	-

Имя прецедента	Изменение сайта
ID	14
Краткое описание	Администратор сайта может изменять сайт и дорабатывать изменения
Актёры, вовлеченные в прецедент	Администратор
Основной поток	<ol style="list-style-type: none"> 1. Прецедент начинается, когда Администратор собирается изменить сайт 2. Администратор подбирает основные изменения, а далее внедряет их в существующую базу страницы сайта
Постусловия	-
Альтернативные потоки	-

Выводы: в ходе выполнения лабораторной работы мы ознакомились с методологией RUP и структурой SRS документа, была создана Use-Case диаграмма и составлен список требований к сайту, оформленный в виде документа SRS.

Вопросы:

1. Методологии разработки ПО. Унифицированный процесс.

Основные модели разработки ПО:

Code and fix — модель кодирования и устранения ошибок;

Waterfall Model — каскадная модель, или «водопад». В этой модели разработка осуществляется поэтапно: каждая следующая стадия начинается только после того, как заканчивается предыдущая. «Водопад» подходит для разработки проектов в *медицинской и космической отрасли, где уже сформирована обширная база документов (СНиПов и спецификаций)*, на основе которых можно написать требования к новому ПО;

V-model — V-образная модель, разработка через тестирование. Это усовершенствованная каскадная модель, в которой заказчик с командой программистов одновременно составляют требования к системе и описывают, как будут тестировать её на каждом этапе. V-модель подходит для проектов, в которых важна надёжность и цена ошибки очень высока. Например, при разработке подушек безопасности для автомобилей или систем наблюдения за пациентами в клиниках.;

Incremental Model — инкрементная модель. Это модель разработки по частям. Инкрементная модель подходит для проектов, в которых точное техзадание прописано уже на старте, а продукт должен быстро выйти на рынок;

Iterative Model — итеративная (или итерационная) модель. Это модель, при которой заказчик не обязан понимать, какой продукт хочет получить в итоге, и может не прописывать сразу подробное техзадание. Итеративная модель подходит для работы над большими проектами с неопределёнными требованиями, либо для задач с инновационным подходом, когда заказчик не уверен в результате.;

Spiral Model — спиральная модель. Используя эту модель, заказчик и команда разработчиков серьёзно анализируют риски проекта и выполняют его итерациями. Последующая стадия основывается на предыдущей, а в конце каждого витка — цикла итераций — принимается решение, продолжать ли проект.;

Chaos model — модель хаоса;

Prototype Model — прототипная модель.

Из этих моделей наиболее популярны пять основных: каскадная, V-образная, инкрементная, итерационная и спиральная.

Rational Unified Process (RUP) — методология разработки программного обеспечения, созданная компанией Rational Software. RUP использует итеративную модель разработки. В конце каждой итерации (в идеале продолжающейся от 2 до 6 недель) проектная команда должна достичь запланированных на данную итерацию целей, создать или доработать проектные артефакты и получить промежуточную, но функциональную версию конечного продукта. Итеративная разработка позволяет быстро реагировать на меняющиеся требования, обнаруживать и устранять риски на ранних стадиях проекта, а также эффективно контролировать качество создаваемого продукта. Первые идеи итеративной модели разработки были заложены в «спиральной модели».

Полный жизненный цикл разработки продукта состоит из четырёх фаз, каждая из которых включает в себя одну или несколько итераций:

1. Начальная стадия (Inception)

В фазе начальной стадии:

- Формируются видение и границы проекта.
- Создается экономическое обоснование (business case).
- Определяются основные требования, ограничения и ключевая функциональность продукта.
- Создается базовая версия модели прецедентов.
- Оцениваются риски.

При завершении начальной фазы оценивается достижение этапа жизненного цикла цели (англ. Lifecycle Objective Milestone), которое предполагает соглашение заинтересованных сторон о продолжении проекта.

2. Уточнение (Elaboration)

В фазе «Уточнение» производится анализ предметной области и построение исполняемой архитектуры. Это включает в себя:

- Документирование требований (включая детальное описание для большинства прецедентов).
- Спроектированную, реализованную и протестированную исполняемую архитектуру.
- Обновленное экономическое обоснование и более точные оценки сроков и стоимости.
- Сниженные основные риски.

Успешное выполнение фазы уточнения означает достижение этапа жизненного цикла архитектуры (англ. Lifecycle Architecture Milestone).

3. Построение (Construction)

В фазе «Построение» происходит реализация большей части функциональности продукта. Фаза Построение завершается первым внешним

релизом системы и вехой начальной функциональной готовности (Initial Operational Capability).

4. Внедрение (Transition)

В фазе «Внедрение» создается финальная версия продукта и передается от разработчика к заказчику. Это включает в себя программу бета-тестирования, обучение пользователей, а также определение качества продукта. В случае, если качество не соответствует ожиданиям пользователей или критериям, установленным в фазе Начало, фаза Внедрение повторяется снова. Выполнение всех целей означает достижение вехи готового продукта (Product Release) и завершение полного цикла разработки.

2. Требования и их категоризация. Атрибуты требований.

Требование — потребность или ожидание, которое установлено, обычно предполагается или является обязательным.

Требования к ПО могут быть:

1. Функциональными и нефункциональными.

- Функциональные требования описывают функции, которые должно выполнять ПО. Например, предоставлять канал коммуникации для пользователя или переводить данные из одного формата в другой. То есть, речь идет о функционале продукта.
- Нефункциональные требования касаются таких вещей, как доступность, надежность, способность к восстановлению, поддерживаемость, масштабируемость, производительность, безопасность и прочие «...ость».

2. Производными и навязанными.

- Вытекает ли это требование из других требований?
- Это требование было явно и недвусмысленно высказано стейкхолдерами?

3. Ориентированными на продукт или на процесс его разработки.

- «Программа должна проверять права пользователя» — это требование к продукту.
- «Программа должна разрабатываться инкрементально. В ходе разработки должна использоваться непрерывная интеграция» — это требование к процессу.

4. Разной приоритетности. Для назначения приоритета может использоваться шкала с фиксированными значениями «обязательно», «крайне желательно», «желательно» и «необязательно».

5. Разного масштаба. Одни требования касаются проектирования отдельных компонентов, другие — архитектуры всего ПО. Нефункциональные требования чаще всего касаются всей программы в целом.

6. Изменчивыми или стабильными. Например, изначально может быть ясно, что требования будут меняться в ходе жизненного цикла продукта. В таком случае реализация программы должна быть толерантной к внесению изменений.

Атрибуты требований – требование в виде объекта, который обладает свойствами, отличающими его от других требований. В дополнение к тестовому описанию, каждое функциональное требование должно иметь несколько поддерживающих его фрагментов информации, или атрибутов (attributes), связанных с ним. Эти атрибуты представляют контекст и основу каждого требования, они располагаются за описанием предполагаемой функциональности. Вы можете сохранить атрибуты в таблице, базе данных или, что наиболее эффективно, в средстве управления требованиями. Сложно управлять более десятком атрибутов, когда они хранятся только в документах. Средства управления требованиями предоставляют несколько сгенерированных системой атрибутов, а также предоставляют возможность определить другие атрибуты, часть из которых может создаваться

автоматически. Эти средства позволяют фильтровать, сортировать выбранные подмножества требований на основании значений их атрибутов и запрашивать базу данных для их просмотра. Например, можно вызвать список всех высокоприоритетных требований, которые Шери должна реализовать в версии 2.3 и которые имеют статус Approved (Одобрено). Далее приведены возможные атрибуты требований:

- дата создания требования;
- номер текущей версии требования;
- автор, создавший требование;
- приоритет;
- состояние требования;
- происхождение или источник требования;
- логическое обоснование требования;
- номер выпуска или итерации, на которую назначено требование;
- контактное лицо или ответственный за принятие решений по внесению изменений в требование;
- метод проверки или критерий приемки.

3. Язык UML.

UML— язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML является языком широкого профиля, это — открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения,

визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода.

4. Прецеденты использования. UseCase-диаграммы - состав, виды связей.

Диаграмма прецедентов или диаграмма вариантов использования (англ. use case diagram) в UML — диаграмма, отражающая отношения между акторами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Прецедент — возможность моделируемой системы (часть её функциональности), благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат. Прецедент соответствует отдельному сервису системы, определяет один из вариантов её использования и описывает типичный способ взаимодействия пользователя с системой. Варианты использования обычно применяются для спецификации внешних требований к системе.

Основное назначение диаграммы — описание функциональности и поведения, позволяющее заказчику, конечному пользователю и разработчику совместно обсуждать проектируемую или существующую систему.

При моделировании прецедентов системный аналитик стремится:

- чётко отделить систему от её окружения;
- определить язык или система программирования) при составлении модели прецедентов опускаются (для них составляется другой документ).

Актёр обозначает любые сущности, использующие систему. Этими сущностями могут быть люди, технические устройства или даже другие системы.

Вариант использования (Эллипс) – набор действий, который может быть использован актёром для взаимодействия с системой.

Отношение ассоциации – отображают на диаграмме информацию о том, какие варианты использования могут быть использованы каждым актёром.

Отношение обобщения означает, что некоторый актёр (вариант использования) может быть обобщён до другого актёра (варианта использования).

Когда мы используем отношение включения, мы подразумеваем, что составные варианты использования обязательно входят в состав общего варианта использования.

Отношение расширения — это выборочное отношение включения. Если отношение включения обозначает, что элемент обязательно включается в состав другого элемента, то в случае отношения расширения - это включение необязательно.

Лабораторная работа №2

Задание:

Сконфигурировать в своём домашнем каталоге репозитории svn и git и загрузить в них начальную ревизию файлов с исходными кодами (в соответствии с выданным вариантом).

Воспроизвести последовательность команд для систем контроля версий svn и git, осуществляющих операции над исходным кодом, приведённые на блок-схеме.

При составлении последовательности команд необходимо учитывать следующие условия:

Цвет элементов схемы указывает на пользователя, совершившего действие (красный - первый, синий - второй).

Цифры над узлами - номер ревизии. Ревизии создаются последовательно.

Необходимо разрешать конфликты между версиями, если они возникают.

Вопросы:

1. Системы контроля версий - назначение, примеры решений.

Системы контроля версий — это программные инструменты, помогающие командам разработчиков управлять изменениями в исходном коде с течением времени. В свете усложнения сред разработки они помогают командам разработчиков работать быстрее и эффективнее. Системы контроля версий наиболее полезны командам DevOps, поскольку помогают сократить время разработки и увеличить количество успешных развертываний.

То есть разработчики могут находиться в разных концах света и работать над одним проектом. Нет Git используют не только разработчики, но и дизайнеры, писатели, редакторы, проектировщики, переводчики. GitHub часто используют hr специалисты и hr менеджеры для поиска успешных кандидатов на те или иные вакансии в различных областях. Для математического анализа успешности проектов. Система Git очень экономична и не требует рассылки большого количества файлов. Отслеживаются и пересылаются изменения в файлах и ссылки на эти изменения. То есть основная рассылка — это рассылка разницы в ваших редактированиях.

2. Ревизии и ветки.

Под веткой принято понимать независимую последовательность коммитов в хронологическом порядке. Однако конкретно в Git реализация ветки выполнена как указатель на последний коммит в рассматриваемой ветке. После создания ветки уже новый указатель ссылается на текущий коммит.

Имя основной ветки Git-проекта по умолчанию — master (однако зачастую бывает main, например, в GitHub), она появляется сразу при инициализации репозитория.

Ревизия — базовое понятие систем контроля версий. Любое зафиксированное изменение в системе контроля версий называется ревизией. Фиксация изменений создаёт ревизию, но сама ревизия может содержать внутри себя либо дельту изменений, либо снимок.

Кстати, процесс переключения между ревизиями также имеет своё название. Когда мы загружаем конкретную ревизию, то говорят, что переключаемся на неё (checkout).

3. Основные операции над данными в системах контроля версий. Основные команды svn и git.

В git:

- `Git init` – создание репозитория.
- `git config --local user.name «название пользователя»` – назначение пользователя.
- `git config --local user.email user1@mail.com` – назначение email.
- `git add .` – Выделение всех объектов.
- `git commit -m «название комита»` – Создание комита
- `git merge «название ветки1» -m «название основной ветки»` – объединение веток.

В svn:

- `svn import «Путь» file:///«Путь» -m «Название комментария»` – создание среды для комитов.
- `svn co file:///«Путь»` – Создание папки со средой в которой делаются комиты.
- `svn add *` – Выделение всех объектов.
- `svn ci -m «Название комита» --username= «Имя пользователя»` – Создание комита с пользователем.
- `svn merge file:///«Путь»` – Объединение всех объектов.
- `svn update` – Обновление папки с комитами.

4. Виды конфликтов и способы их решения.

Git прерывает работу в самом начале слияния (error: Entry '<fileName>' not uptodate. Cannot merge. (Changes in working directory))

Выполнение команды слияния прерывается в самом начале, если Git обнаруживает изменения в рабочем каталоге или разделе проиндексированных файлов текущего проекта. Git не может выполнить слияние, поскольку иначе эти ожидающие изменения будут перезаписаны новыми коммитами. Такое случается из-за конфликтов не с другими разработчиками, а с ожидающими локальными изменениями. Локальное состояние необходимо стабилизировать с помощью команд `git stash`, `git checkout`, `git commit` или `git reset`.

Git прерывает работу во время слияния (error: Entry '<fileName>' would be overwritten by merge. Cannot merge. (Changes in staging area))

Сбой В ПРОЦЕССЕ слияния говорит о наличии конфликта между текущей локальной веткой и веткой, с которой выполняется слияние. Это свидетельствует о конфликте с кодом другого разработчика. Git сделает все возможное, чтобы объединить файлы, но оставит конфликтующие участки, чтобы вы разрешили их вручную.

Самый простой способ разрешить конфликт — отредактировать конфликтующий файл. Откройте файл merge.txt в привычном редакторе. Измененное содержимое файла merge.txt будет выглядеть следующим образом:

```
this is some content to mess with  
  
content to append  
  
totally different content to merge later
```

После редактирования файла выполните команду `git add merge.txt`, чтобы добавить новое объединенное содержимое в раздел проиндексированных файлов. Для завершения слияния создайте новый коммит, выполнив следующую команду:

```
git commit -m "merged and resolved the conflict in merge.txt"
```

Git обнаружит, что конфликт разрешен, и создаст новый коммит слияния для завершения процедуры слияния.

Команды Git, с помощью которых можно разрешить конфликты слияния.

`git status` – команда `status` часто используется во время работы с Git и помогает идентифицировать конфликтующие во время слияния файлы.

`git log –merge` – при передаче аргумента `--merge` для команды `git log` будет создан журнал со списком конфликтов коммитов между ветками, для которых выполняется слияние.

`git diff` – команда `diff` помогает найти различия между состояниями репозитория/файлов. Она полезна для выявления и предупреждения конфликтов слияния.

Инструменты для случаев, когда Git прерывает работу в самом начале слияния

`git checkout` – команда `checkout` может использоваться для отмены изменений в файлах или для изменения веток.

`git reset –mixed` – команда `reset` может использоваться для отмены изменений в рабочем каталоге или в разделе проиндексированных файлов.

Инструменты для случаев, когда конфликты Git возникают во время слияния

`git merge –abort` – при выполнении команды `git merge` с опцией `--abort` процесс слияния будет прерван, а ветка вернется к состоянию, в котором она находилась до начала слияния.

`git reset` – команду `git reset` можно использовать для разрешения конфликтов, возникающих во время выполнения слияния, чтобы восстановить заведомо удовлетворительное состояние конфликтующих файлов.

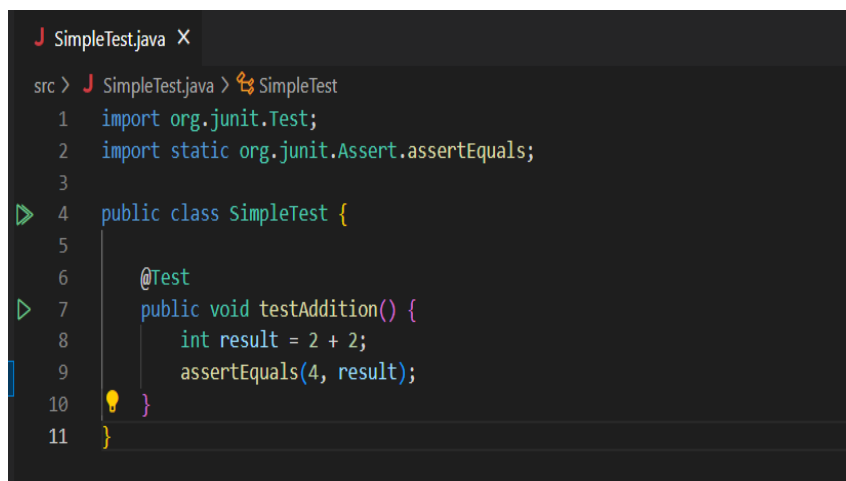
Лабораторная работа №3

Задание:

Написать сценарий для утилиты Apache Ant, реализующий компиляцию, тестирование и упаковку в jar-архив кода проекта из лабораторной работы №3 по дисциплине "Веб-программирование".

Каждый этап должен быть выделен в отдельный блок сценария; все переменные и константы, используемые в сценарии, должны быть вынесены в отдельный файл параметров; MANIFEST.MF должен содержать информацию о версии и о запускаемом классе.

Пример Junit теста на java:



```
J SimpleTest.java X
src > J SimpleTest.java > SimpleTest
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3
4  public class SimpleTest {
5
6      @Test
7      public void testAddition() {
8          int result = 2 + 2;
9          assertEquals(4, result);
10     }
11 }
```

Build код

```
1 <project name="MyProject" default="build">
2   <property name="src.dir" value="src"/>
3   <property name="build.dir" value="build"/>
4   <property name="dist.dir" value="dist"/>
5   <property name="main-class" value="MyMainClass"/>
6
7   <target name="init">
8     <mkdir dir="${build.dir}"/>
9     <mkdir dir="${dist.dir}"/>
10  </target>
11
12  <target name="compile" depends="init">
13    <javac srcdir="${src.dir}" destdir="${build.dir}"/>
14  </target>
15
16  <target name="test" depends="compile">
17    <junit printsummary="true" haltonfailure="true">
18      <classpath>
19        <pathelement location="${build.dir}"/>
20      </classpath>
21      <formatter type="plain"/>
22      <test name="MyTestClass"/>
23    </junit>
24  </target>
25
26  <target name="jar" depends="compile">
27    <jar destfile="${dist.dir}/myproject.jar" basedir="${build.dir}">
28      <manifest>
29        <attribute name="Main-Class" value="${main-class}"/>
30      </manifest>
31    </jar>
32  </target>
33
34  <target name="build" depends="test,jar">
35    <echo message="Build completed"/>
36  </target>
37 </project>
```

Выполнение

```
C:\avt\JUnit>ant init
ANT_OPTS is set to -Djava.security.manager=allow
Buildfile: C:\avt\JUnit\build.xml

init:
    [mkdir] Created dir: C:\avt\JUnit\dist

BUILD SUCCESSFUL
Total time: 0 seconds

C:\avt\JUnit>_
```

Вывод:

При выполнении данной лабораторной работы были изучены основы работы с Apache Ant и написаны первые junit-тесты.

Вопросы:

1. Тестирование ПО. Цель тестирования, виды тестирования.

Тестирование программного обеспечения — процесс исследования программного обеспечения (ПО) с целью получения информации о качестве продукта.

Цели тестирования: Обнаружение дефектов Повышение уверенности в уровне качества Предоставление информации для принятия решений Предотвращение дефектов.

Виды тестирования:

- Функциональное и нефункциональное

Функциональное тестирование — это проверка функций программы. Специалист нажимает на всевозможные клавиши и пытается вести себя необычно, дабы обнаружить недочеты проекта.

Нефункциональное тестирование представляет собой проверку производительности, надежности и отзывчивости приложения, а также ее соответствия нормам безопасности.

- Статическое и динамическое

Статическая проверка выполняется с выключенной программой. Специалисты открывают документацию приложения, анализируют указанные в ней функции, а затем изучают код для оценки качества реализации.

Динамическое тестирование выполняется после статического. В этом случае необходимо включить программу и на практике узнать, насколько работоспособными являются ее функции.

- Нагрузочное. Речь идёт о тестировании программы в условиях высоких нагрузок, которые могут быть больше, чем планировали

разработчики. Эти тесты обязательны для онлайн-сервисов, которые должны правильно работать даже при наличии большого числа посетителей на пиковой или регулярной основе (онлайн-магазины во время распродаж, новостные ресурсы при резонансных событиях и т.д.).

- Тестирование UX. В этом случае специалист сосредотачивается на пользовательском опыте. Тестировщику необходимо поставить себя на место клиента. На основе составленных им замёток в процессе взаимодействия с приложением будут вноситься соответствующие изменения.

- Конфигурационное. Это проверка совместимости программы с аппаратным обеспечением и прочими software-элементами (различными версиями OS и процессоров). Конфигурационное тестирование необходимо для межплатформенных программ и в процессе перехода поставщика платформы на принципиально новую аппаратную базу (яркий пример — появление ноутбуков с чипами M1 от Apple).

2. Модульное тестирование, основные принципы и используемые подходы.

Модульное тестирование — это процесс проверки функциональности отдельных модулей программного обеспечения. Модуль — это независимый компонент программы, который может быть протестирован отдельно от других модулей.

В модульном тестировании программисты создают тестовые сценарии для каждого модуля, которые проверяют корректность его работы. Если тест не проходит, программисты находят и исправляют ошибки до тех пор, пока тест не будет пройден успешно.

Существует множество инструментов для модульного тестирования, таких как JUnit, NUnit, PHPUnit и другие. Они обеспечивают возможность создания тестовых сценариев и автоматического выполнения тестов.

«Ящичный» подход. Согласно этому подходу, все тесты ПО делятся на три вида ящиков:

- Тестирование типа «белый ящик» проверяет внутренние структуры и модули, игнорирует ожидаемую функциональность для конечных пользователей. Это может быть тестирование API, внесение неисправностей (fault injection), модульное тестирование, интеграционное тестирование.
- Тестирование типа «чёрный ящик» больше интересуется тем, что делает ПО, а не как делает. Это означает, что тестировщики не обязаны ни разбираться в объекте тестирования, ни понимать, как он работает под капотом. Такой тип тестирования нацелен на конечных пользователей, их опыт взаимодействия с видимым интерфейсом. К «чёрным ящикам» относится тестирование на основе моделей, тестирование способов использования, таблицы переходов состояний, спецификационное тестирование и т. д.
- Тестирование типа «серый ящик» проектируется со знанием программных алгоритмов и структур данных (белый ящик), но выполняется на пользовательском уровне (чёрный ящик). Сюда относится регрессионное тестирование и шаблонное тестирование (pattern testing).

3. Пакет JUnit, основные API.

JUnit — наиболее широко используемая среда тестирования для приложений Java. JUnit долгое время отлично справлялся со своей задачей.

`org.junit.jupiter.api` – JUnit Jupiter API для написания тестов.

`org.junit.jupiter.api.condition` – Условия включения или отключения тестов в JUnit Jupiter на основе аннотаций.

`org.junit.jupiter.api.extension` – JUnit Jupiter API для написания расширений.

org.junit.jupiter.api.function – Функциональные интерфейсы, используемые в JUnit Jupiter.

org.junit.jupiter.api.io – Поддержка ввода-вывода в JUnit Jupiter.

org.junit.jupiter.api.parallel – JUnit Jupiter API для влияния на параллельное выполнение тестов.

4. Системы автоматической сборки. Назначение, принципы работы, примеры систем.

Gradle — система автоматической сборки, которую используют для упрощения работы с Java. С помощью (условно) стандартизированных средств она помогает разработчикам собрать нужный продукт без потери его уникальности.

High-level архитектура. Архитектура всех build-систем сводится к следующему:

- конфигурации
- собственная конфигурация, где хранятся «личные» настройки системы. Например, такие как информация о месте установки или окружении, информация о репозиториях и прочее;
- конфигурация модуля, где описывается место расположения проекта, его зависимости и задачи, которые требуется выполнять для проекта;
- парсеры конфигураций
- парсер способный «прочитать» конфигурацию самой системы, для её настройки соответствующим образом;
- парсер конфигурации модуля, где некоторыми «понятными человеку» терминами описываются задачи для системы сборки;

- сама система — некоторая утилита + скрипт для её запуска в вашей ОС, которая после чтения всех конфигураций начнет выполнять тот или иной алгоритм, необходимый для реализации запущенной задачи;
- система плагинов — дополнительные подключаемые надстройки для системы, в которых описаны алгоритмы реализации типовых задач;
- локальный репозиторий — репозиторий, расположенный на локальной машине, для кэширования запрашиваемых файлов на удаленных репозиториях.

Подробнее об этом вы можете прочитать в инструкции к плагинам: `JavaCompile` (добавить новые задачи), `SourceSet` (добавить новые объекты домена). Также плагины работают с соглашениями и расширяют объекты. Например, с помощью плагина можно добавить новые элементы DSL и (или) настроить их.

Набор "Core Plugins" автоматически создаётся при установке Gradle. На первых этапах рекомендуют использовать категорию "Utility", а именно — плагин "Build Init Plugin". Он предоставляет задачи для инициализации проекта в системе.

После выбора задачи и её успешного завершения вы увидите внизу зелёную надпись "BUILD SUCCESSFUL". Это значит, что (вам повезло) всё прошло без проблем. Также внизу система выдаст краткий отчёт.

Если статус "executed" — задача действительно выполнена. Если "up-to-date" — нет. Это не значит, что произошёл какой-то сбой. В случае такого статуса задача не требует решения в принципе, т. е. её объект уже в актуальном состоянии.

Это произошло потому, что в Gradle автоматически формируется "Up-to-date checks" — инкрементальный билд, цель которого — оптимизация

работы системы. Поэтому задачи, которые уже завершены или не требуют действий, не прорабатываются.

Отключить этот build можно. Вручную. Для этого необходимо выполнить задачу с указанием флага `--rerun-tasks`. Если всё сделано правильно, статус у всех задач изменится на "executed".

Также Gradle позволяет просматривать логи разного уровня. Например, команда `gradle run -i` позволит читать информационные сообщения о работе системы, команда `run -q` — включить режим тишины, `run -d` — показывать все сообщения журнала и т. д. Полный список логов и дополнительная информация о системе логирования тут.

5. Утилита make. Make-файлы, цели и правила.

Утилита make предназначена для автоматизации сборки проектов. Если какие-либо файлы проекта могут быть сгенерированы из других файлов, утилита позволяет выполнить процесс построения наиболее оптимальным способом, по возможности минимизируя количество обрабатываемых файлов.

На сегодняшний день наиболее распространены три варианта утилиты, объединенные общими принципами работы, но отличающиеся синтаксисом языка и возможностями:

- GNU make — самый распространенный и функциональный вариант
- BSD make (pmake) — используется в проектах BSD, по функциональности примерно соответствует GNU make
- nmake (Microsoft make) — работает под Windows, малофункционален, только базовые принципы make.

Утилита make работает по правилам (rules), записанным в специальном конфигурационном файле. Правила определяют цели (targets), зависимости между целями и набор команд для выполнения каждой цели.

Правило make это ответы на три вопроса:

{Из чего делаем? (реквизиты)} ---> [Как делаем? (команды)] ---> {Что делаем? (цели)}

Цели могут соответствовать определенным файлам. Кроме того, цели могут не соответствовать ни одному файлу и использоваться для группировки других целей или определенной последовательности команд. Такие цели называются phony targets.

Каждая цель может зависеть от выполнения других целей. Выполнение цели требует предварительного выполнения других целей, от которых она зависит.

В случае зависимости между целями, соответствующими файлам, цель выполняется только в том случае, если файлы, от которых она зависит, новее, чем файл, соответствующий цели. Это позволяет регенерировать только файлы, зависящие от измененных файлов, и не выполнять потенциально долгий процесс пересборки всех файлов проекта.

Таким образом, makefile определяет граф зависимостей, по которому утилита make выполняет ту или иную цель, по возможности минимизируя количество операций сборки.

6. Утилита Ant. Сценарии сборки, цели и команды.

Ant - это мощный платформно-независимый скриптовый инструмент, используемый для сборки приложений. Сценарий сборки приложения java оформляется в виде XML-файла подобно скриптам "make" при обработке файлов C/C++. По-умолчанию сценарий сборки извлекается из файла build.xml.

Пример описания сценария сборки:

- копирование *.jar файлов в каталог релиза, но перед этим необходимо
- сформировать *.jar файлы, но перед этим необходимо
- скомпилировать java-файлы в файлы *.class

Сценарий сборки ant'у показывает что надо делать, чтобы превратить из того, что есть (как правило, исходный java-код) в то, что необходимо. Сценарий представляет собой детальный план сборки из частей единого целого, включающий ряд операндов, позволяющих выполнять команды копирования, удаления и перемещения файлов, компиляции java-файлов, формирование документации к коду и исполняемого jar-файла. Корневой элемент сценария project может содержать три необязательных атрибута:

- name - имя проекта;
- default - цель проекта по умолчанию;
- basedir - базовая директория, относительно которой будут вычисляться все пути.

Элемент, описывающий цель проекта target может содержать следующие атрибуты:

- name - имя цели, обязательный атрибут;
- depends - промежуточные цели, от которых зависит данная цель; имена перечисляются через запятую;
- if - определяет какие свойства должны быть равны true для запуска цели;
- unless - определяет какие свойства должны быть равны false для запуска цели;

- `description` - краткое описание цели, что она делает.

Параметр `property` определяет пару имя/значение, которая может многократно использоваться в сценарии подобно переменным. Свойства (настройки) можно определять как внутри `build.xml` файла, так и в отдельных файлах. При определении внутри `xml` файла свойства могут включать следующие атрибуты:

- `name` - имя свойства;
- `value` - значение свойства;
- `location` - устанавливает значение свойства в абсолютный путь. Если значение уже абсолютный путь, то ничего не меняется, если относительный, то подставляется базовая директория. Символы `/` и `\` меняются автоматически в зависимости от платформы;
- `refid` - ссылка на другой объект, определенный где-либо;
- `resource` - имя ресурса содержащего настройки в формате настроечного файла;
- `file` - путь к файлу настройки (в нем свойства определяются как имя=значение на отдельной строке);
- `url` - адрес настройки;
- `environment` - префикс используемый для доступа к переменным окружения. Например, если определено `myenv`, то к переменным обращаются как `"myenv.PATH"`;
- `classpath` - путь к ресурсам;
- `prefix` - префикс добавляемый к свойствам загруженных из файла, ресурса, или `url`. По умолчанию префикс `"."`.

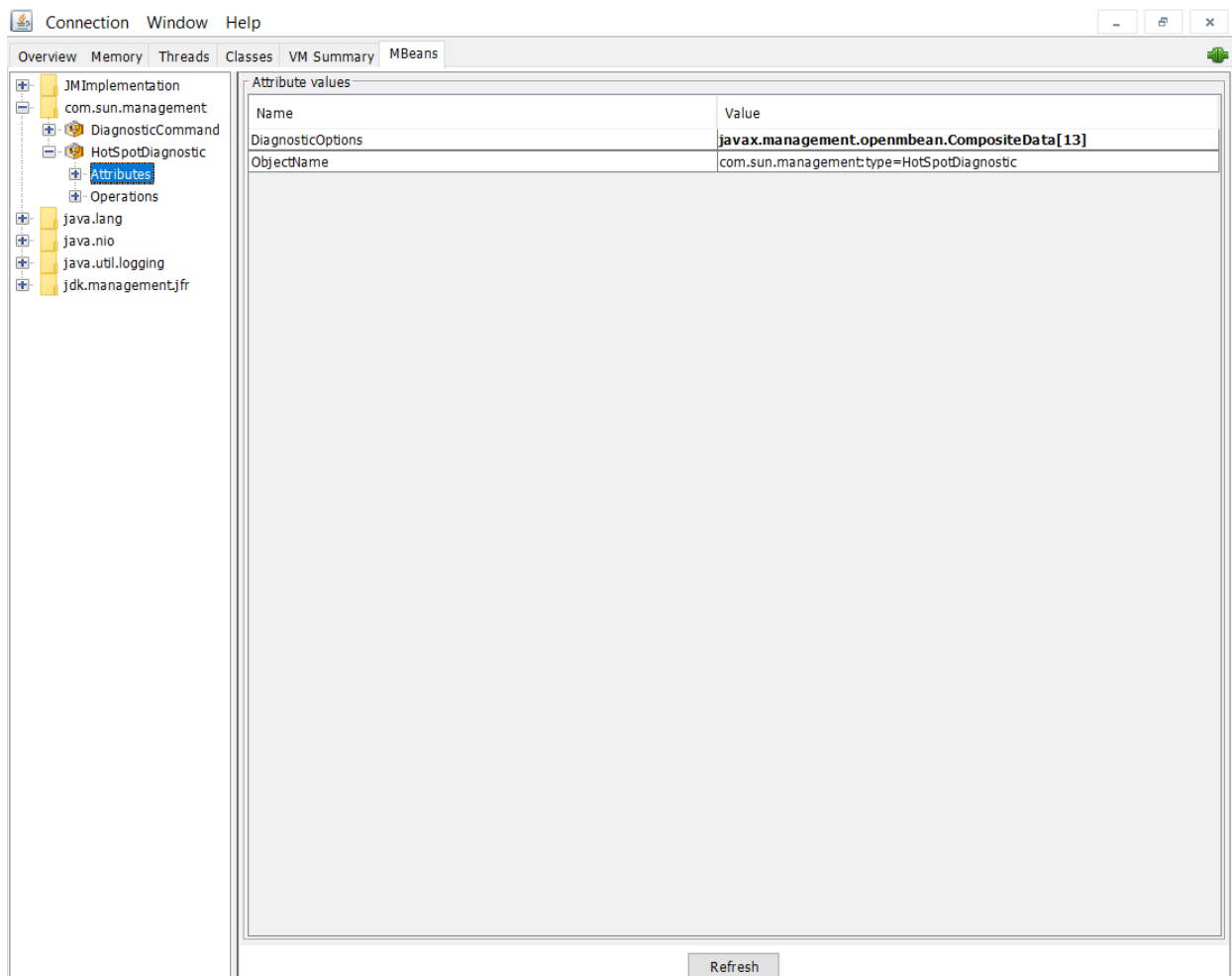
Лабораторная работа №4

MBean пример кода:

```
1  public class ClicksPercentageMonitor implements ClicksPercentageMonitorMBean {
2      private int totalClicks;
3      private int hits;
4
5      public ClicksPercentageMonitor() {
6          totalClicks = 0;
7          hits = 0;
8      }
9
10     public void clicked(int x, int y) {
11         totalClicks++;
12         if (isHit(x, y)) {
13             hits++;
14         }
15     }
16
17     public double getPercentage() {
18         return (double) hits / totalClicks * 100;
19     }
20
21     private boolean isHit(int x, int y) {
22         // Check if the coordinates (x,y) are within the target area
23         // on the coordinate plane where we want to measure the clicks
24         // and return true if they are.
25     }
26 }
27
28 public interface ClicksPercentageMonitorMBean {
29     double getPercentage();
30 }
```

Показания MBean-классов:

2. JConsole



The screenshot shows the JConsole application window with the 'MBeans' tab selected. The left sidebar displays a tree of MBean classes, including 'JMImplementation', 'com.sun.management', 'DiagnosticCommand', 'HotSpotDiagnostic', 'Attributes', 'Operations', 'java.lang', 'java.nio', 'java.util.logging', and 'jdk.management.jfr'. The 'Attributes' node under 'HotSpotDiagnostic' is selected. The main pane shows a table of attribute values for the selected MBean.

Name	Value
DiagnosticOptions	<code>javax.management.openmbean.CompositeData[13]</code>
ObjectName	<code>com.sun.management:type=HotSpotDiagnostic</code>

Refresh

Java Monitoring & Management Console - pid: 11816 jdk.jconsole/sun.tools.jconsole.JConsole localhost:10999

Connection Window Help

OverviewMemoryThreadsClassesVM SummaryMBeans

+

JMImplementation

+

com.sun.management

+

DiagnosticCommand

+

HotSpotDiagnostic

+

Attributes

+

Operations

+

java.lang

+

java.nio

+

java.util.logging

+

jdk.management.jfr

Operation invocation

void

dumpThreads

(

p0

String

,

p1

String

)

void

dumpHeap

(

p0

String

,

p1

true

)

CompositeData

getVMOption

(

p0

String

)

void

setVMOption

(

p0

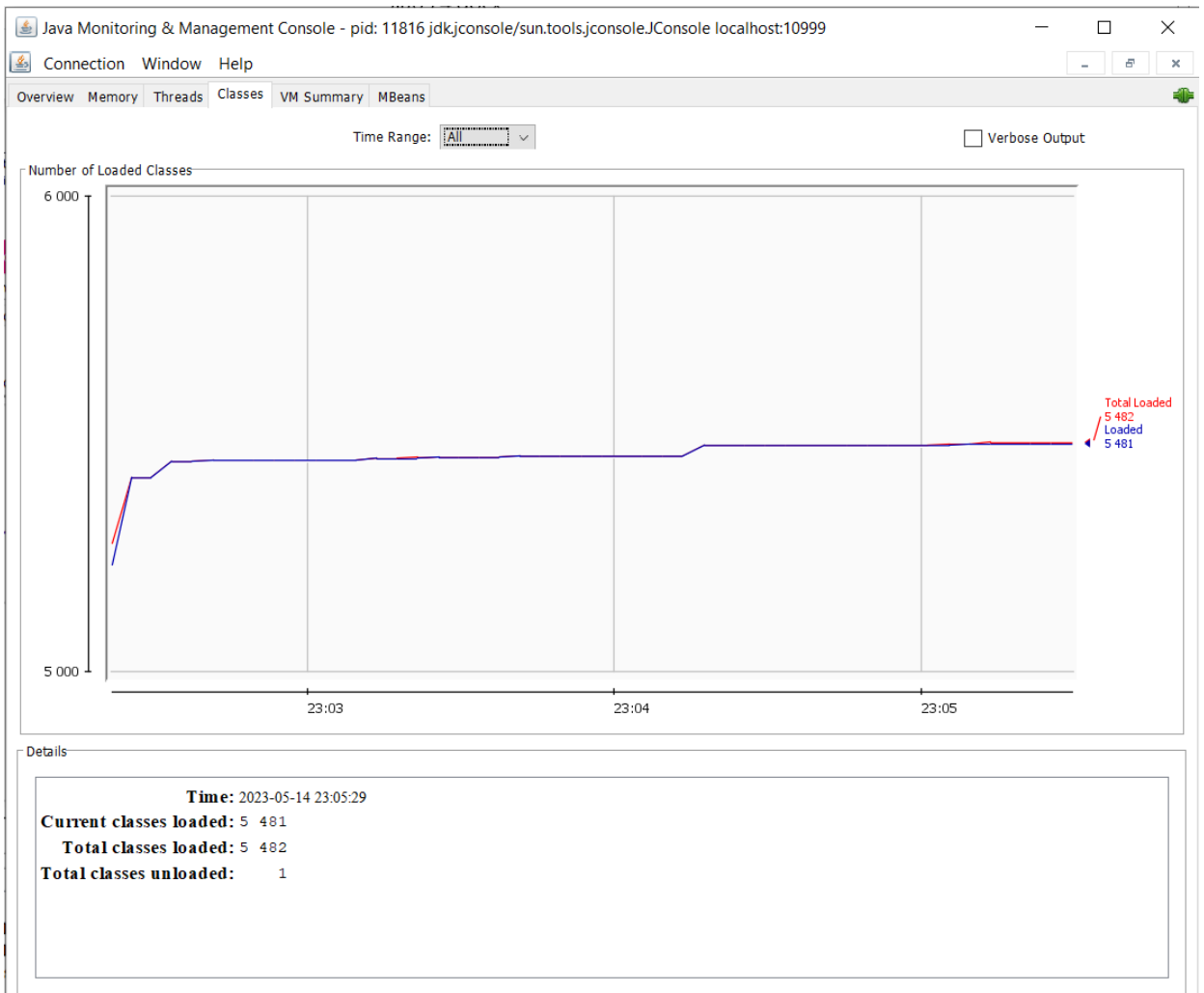
String

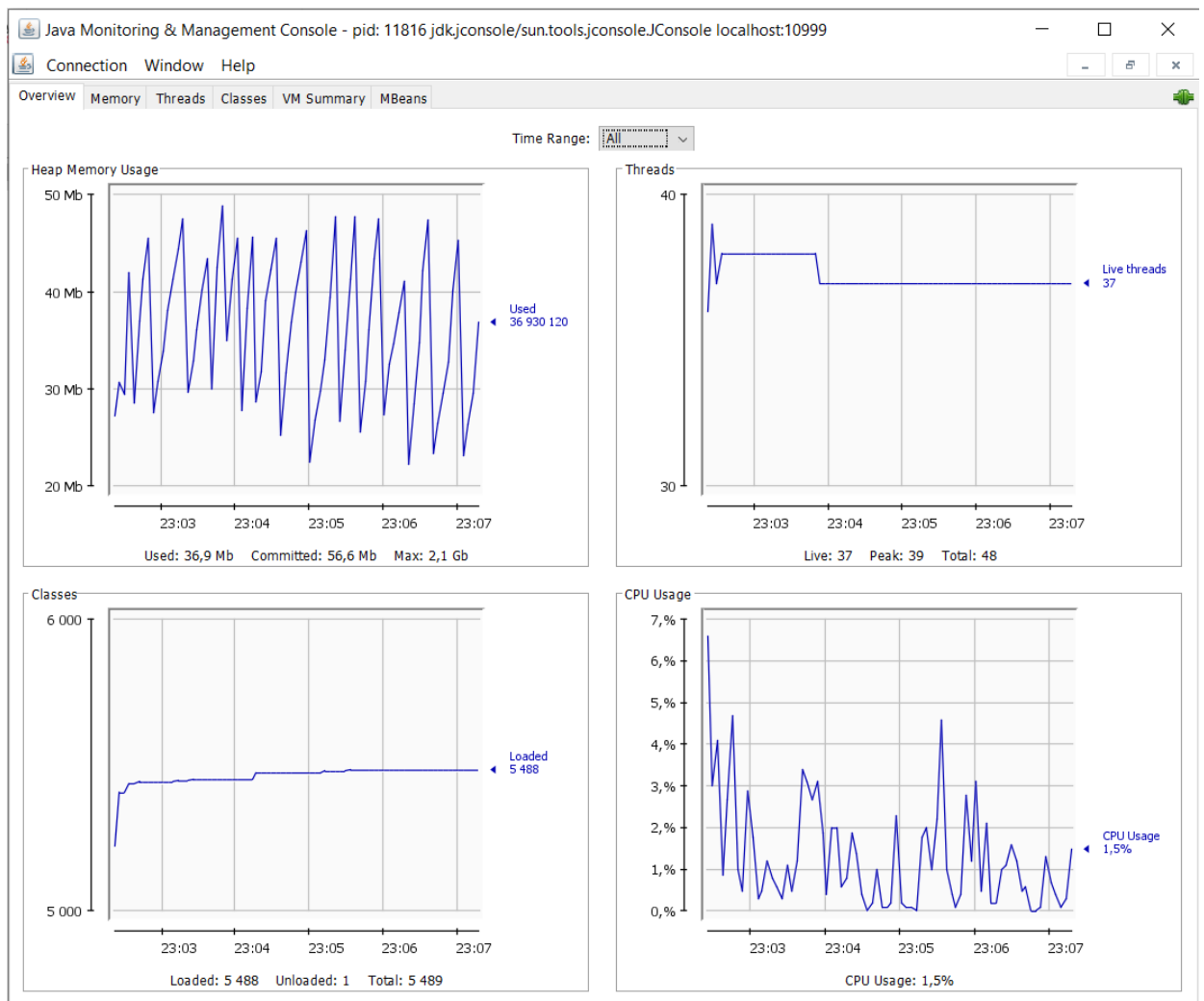
,

p1

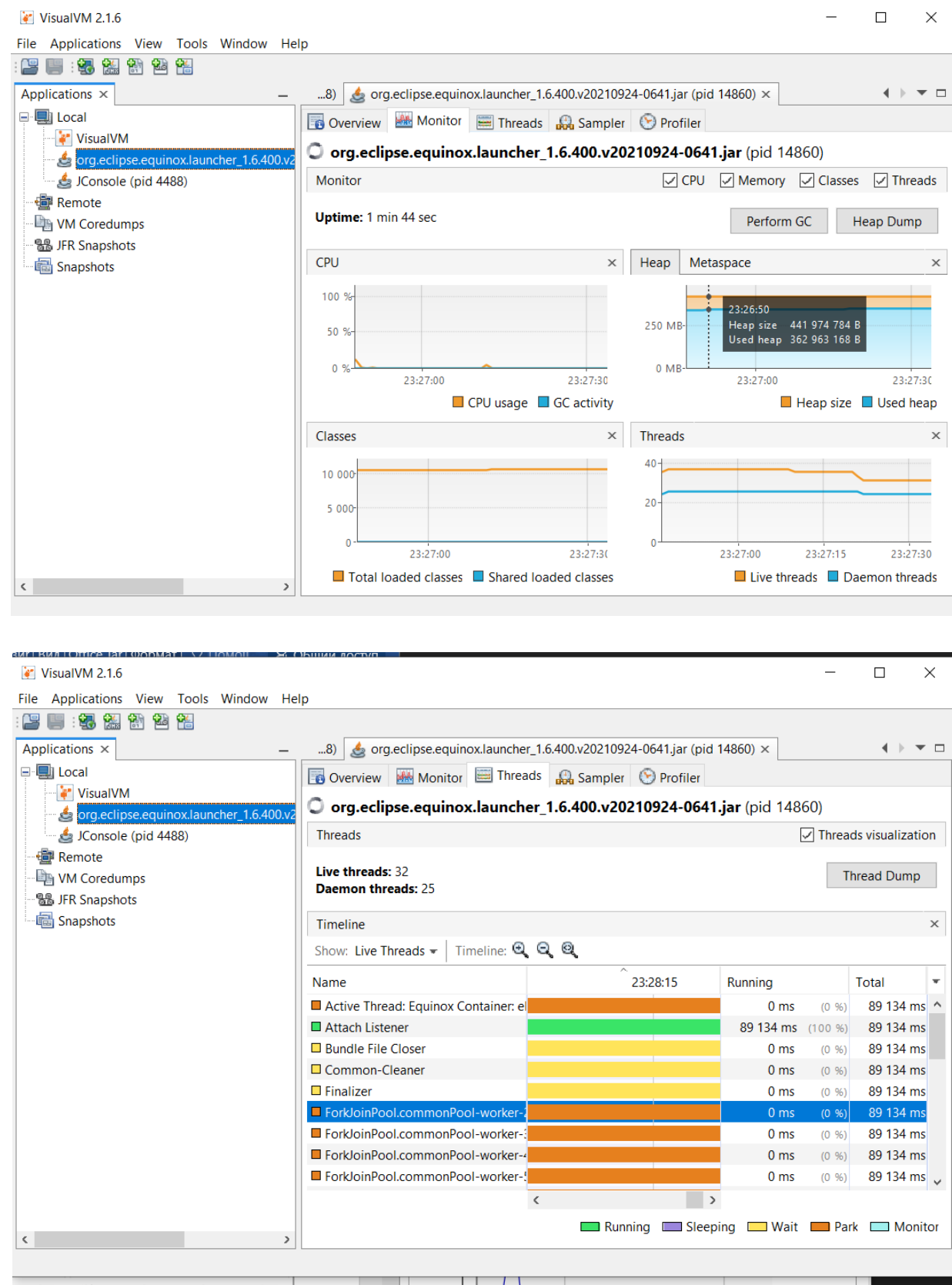
String

)





3. VisualVM



VisualVM 2.1.6

File Applications View Tools Window Help

Applications x ...8) org.eclipse.equinox.launcher_1.6.400.v20210924-0641.jar (pid 14860) x

Overview Monitor Threads Sampler Profiler

org.eclipse.equinox.launcher_1.6.400.v20210924-0641.jar (pid 14860)

Sampler Settings

Sample: CPU Memory Stop

Status: CPU sampling in progress

CPU samples Thread CPU time

Results: View: Collected data: Snapshot Thread Dump

Name	Total Time	Total Time (CPU)
main	4 097 ms (100 %)	0,0 ms (- %)
Framework Event Dispatcher: Equinox Conta	4 097 ms (100 %)	0,0 ms (- %)
Start Level: Equinox Container: eb9d5995-c1	4 097 ms (100 %)	0,0 ms (- %)
SCR Component Actor	4 097 ms (100 %)	0,0 ms (- %)
Worker-JM	4 097 ms (100 %)	0,0 ms (- %)
Worker-0	4 097 ms (100 %)	0,0 ms (- %)
Worker-1	4 097 ms (100 %)	0,0 ms (- %)
Java indexing	4 097 ms (100 %)	0,0 ms (- %)
pool-1-thread-1	4 097 ms (100 %)	4 097 ms (100 %)
Bundle File Closer	4 097 ms (100 %)	0,0 ms (- %)

VisualVM 2.1.6

File Applications View Tools Window Help

Applications x ...8) org.eclipse.equinox.launcher_1.6.400.v20210924-0641.jar (pid 14860) x

Overview Monitor Threads Sampler Profiler

org.eclipse.equinox.launcher_1.6.400.v20210924-0641.jar (pid 14860)

Sampler Settings

Sample: CPU Memory Stop

Status: memory sampling in progress

Heap histogram Per thread allocations

Results: View: Collected data: Snapshot Perform GC Heap Dump

Name	Live Bytes	Live Objects
byte[]	162 848 216 B (44,1 %)	676 739 (15,7 %)
char[]	75 397 328 B (20,4 %)	532 388 (12,3 %)
char[][]	31 914 784 B (8,7 %)	317 176 (7,4 %)
java.lang.Object[]	19 260 640 B (5,2 %)	438 904 (10,2 %)
java.lang.String	15 390 120 B (4,2 %)	641 255 (14,9 %)
java.util.TreeMap\$Entry	10 756 720 B (2,9 %)	268 918 (6,2 %)
org.eclipse.jdt.internal.core.util.SimpleWi	6 796 464 B (1,8 %)	283 186 (6,6 %)
com.google.gson.internal.LinkedTreeMa	5 388 192 B (1,5 %)	112 254 (2,6 %)
int[]	4 163 344 B (1,1 %)	11 092 (0,3 %)
java.lang.String[]	3 400 128 B (0,9 %)	106 093 (2,5 %)
java.util.HashMap\$Node	2 259 808 B (0,6 %)	70 619 (1,6 %)

Вывод:

При выполнении данной лабораторной работы был написан MBean и проведено ознакомление с JConsole и VisualVM

Вопросы:

1. Мониторинг и профилирование. Основные понятия. Отличия мониторинга от профилирования.

Мониторинг — система постоянного наблюдения за явлениями и процессами, проходящими в окружающей среде и обществе, результаты которого служат для обоснования управленческих решений по обеспечению безопасности людей и объектов экономики.

Профилирование — это сбор характеристик программы во время ее выполнения. При профилировании замеряется время выполнения и количество вызовов отдельных функций и строк в коде программы. При помощи этого инструмента программист может найти наиболее медленные участки кода и провести их оптимизацию.

Мониторинг = отслеживание того, как запущено приложение (использование памяти, процессора, диска и т.д.). Профилирование = измерение производительности (скорости) путем запуска кода и определения времени, которое занимает каждый шаг.

2. Инфраструктура для организации мониторинга и профилирования в составе JDK. JMX.

Начиная с 7u40 в составе Oracle JDK появился GUI инструмент для реалтайм мониторинга и профилирования JVM. Он позволяет собрать очень много информации о JVM процессе, найти узкие места и проблемы, которые влияют на производительность.

Mission Control включает в себя JMX консоль и Java Flight Recorder (JFR). С помощью JMX осуществляется взаимодействие с удаленным Java процессом, а JFR собирает данные о событиях.

3. MBeans. Основные понятия. Архитектура фреймворка.

MBEANS технологии JMX - это managed beans, а именно объекты Java, которые представляют ресурсы, подлежащие управлению. MBean имеет management interface, состоящий из следующего:

- Именованные и типизированные атрибуты, которые могут быть прочитаны и записаны.
- Именованные и типизированные операции, которые могут быть вызваны.
- Типизированные уведомления, которые могут быть отправлены MBean.

Платформа MXBean - это MBean для мониторинга виртуальной машины Java и других компонентов среды выполнения Java и управления ими. Каждый MXBean инкапсулирует часть функциональных возможностей виртуальной машины, таких как система загрузки классов, система компиляции точно в срок (JIT), сборщик мусора.

Каждый экземпляр WebLogic владеет сервером MBean, на котором размещено несколько MBeans. Сервер MBean действует как реестр для MBeans и предоставляет службы для доступа к MBeans, запущенным на сервере, и управления ими. Поскольку домен WebLogic может быть распределен по нескольким компьютерам с различными развертываниями и ресурсами домена, предназначенными для разных серверов, сервер MBean для каждого экземпляра WebLogic будет содержать разные типы MBeans. Например, статистику времени выполнения для пула подключений JDBC на сервере можно получить только из MBean, запущенного на этом сервере.

4. Утилита JConsole. Возможности, область применения.

Графический пользовательский интерфейс JConsole - это инструмент мониторинга, который соответствует спецификации Java Management Extensions (JMX). JConsole использует обширный инструментарий

виртуальной машины Java (Java VM) для предоставления информации о производительности и потреблении ресурсов приложениями, работающими на платформе Java. Можно легко использовать JConsole (или его ближайший родственник VisualVM) для мониторинга производительности приложений Java и отслеживания кода в Java.

5. Утилита Visual VM. Возможности, область применения.

Java VisualVM - это инструмент, который предоставляет визуальный интерфейс для просмотра подробной информации о приложениях, основанных на технологии Java (Java applications), во время их запуска на виртуальной машине Java (JVM). Java VisualVM организует данные JVM, которые извлекаются средствами Java Development Kit (JDK), и представляет информацию таким образом, чтобы вы могли быстро просматривать данные в нескольких приложениях Java. Вы можете просматривать данные о локальных приложениях и приложениях, запущенных на удаленных хостах. Вы также можете собирать данные о программном обеспечении JVM и сохранять их в своей локальной системе, а затем просматривать данные позже или делиться ими с другими пользователями.

Если у вас уже есть приложение, которое надо оптимизировать — то как пользоваться этими инструментами более-менее понятно.

Если же ваша цель более академическая — то вам понадобится какая-то программа на Java. Начать исследовать можно с этой, сейчас она позволяет создавать в памяти куски памяти и «забывать» о них (создавая «мусор»).

Однако, чтобы хорошо поиграть с виртуальной машиной, надо побольше узнать об особенностях ее работы. Например, можно исследовать следующее:

- написать программу, нарушающую «гипотезу о потоковой локальности» (ведь JVM считает, что большинство объектов используются только одним потоком);

- работу с объектами разного размера — в JVM есть настройки, позволяющие большим объектам сразу попадать в Permanent generation;
- освобождение памяти из-под объектов с финализаторами. Не секрет, что JVM не спешит удалять их, но можно также поместить в финализаторы код, требующий длительных вычислений;
- использование гипотезы о том, что «объекты умирают молодыми».
- работу JIT-компилятора — постоянно перестраивать common path;li>
- написать очень большую программу (скорее всего сгенерировать), исследовать сколько памяти расходуется при оптимизации кода.

6. Удалённый мониторинг и профилирование приложений на платформе Java.

Профилировщик Java - это инструмент, который отслеживает конструкции и операции байт-кода Java на уровне JVM. Эти конструкции и операции кода включают создание объекта, итеративные выполнения (включая рекурсивные вызовы), выполнение методов, выполнение потоков и сборку мусора.

JProfiler - лучший выбор для многих разработчиков. Благодаря интуитивно понятному пользовательскому интерфейсу JProfiler предоставляет интерфейсы для просмотра производительности системы, использования памяти, потенциальных утечек памяти и профилирования потоков.

YourKit обладает основными функциями для визуализации потоков, сборки мусора, использования памяти и утечек памяти с поддержкой локального и удаленного профилирования через ssh-туннелирование.

Java VisualVM - это упрощенный, но надежный инструмент профилирования для приложений Java. Это бесплатный профилировщик с открытым исходным кодом. Этот инструмент входил в комплект Java Development Kit (JDK) вплоть до JDK 8, но был удален в JDK 9 и теперь распространяется как отдельный инструмент: загрузка VisualVM. Его работа зависит от других автономных инструментов, предоставляемых в JDK, таких как JConsole, jstat, jstack, jinfo и jmap.

Профилировщик NetBeans поставляется в комплекте с IDE Oracle NetBeans с открытым исходным кодом. Хотя этот профилировщик имеет много общего с Java VisualVM, это хороший выбор, когда мы хотим, чтобы все было объединено в одной программе (IDE + Profiler).

IntelliJ Profiler - это простой, но мощный инструмент для профилирования распределения ресурсов процессора и памяти. Он сочетает в себе возможности двух популярных профилировщиков Java: JFR и Async profiler.

Несмотря на наличие некоторых расширенных функций, основное внимание уделяется простоте использования. IntelliJ Profiler позволяет нам начать работу в несколько кликов без какой-либо настройки, предоставляя полезные функции, помогающие в нашей повседневной работе по разработке.