МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

Отчет по практической работе №5

по дисциплине «**Тестирование и верификация ПО**»

Выполнил:
Студент групп ИКБО-04-21 Даурбеков М.И.

Принял руководитель работы: Петренко А.А.

Практические работы выполнены

Зачтено 9.12.2023

2023 г.

СОДЕРЖАНИЕ

1. Цель работы 3
2. Теоретическое введение 3
3. Ход работы 6
Выводы 10
Список использованных источников 12

1. Цель работы

Получить первоначальные навыки тестирования с помощью статического и динамического анализаторов кода

1.1. Задание

Часть 1

1. Проверить ранее сделанный учебный проект несколькими статическим анализатором (от 3-5). Сделать вывод о адекватности найденных ошибок.
2. Внести разные типы ошибок и проверить работу анализаторов
3. Сделать вывод о целесообразности статического анализа.
4. Предложить варианты написания своего анализатора для решенияпроблем из своего опыта, которые возникали слишком часто или имели негативные последствия. Реализованы ли такие средства в текущих анализаторах, что Вы попробовали?
5. Проанализировать учебный код динамическим анализатором.
6. Внести ошибки и проверить адекватность работы динамического анализатора.
7. Оценить возможность создания автоматной модели по коду пример.

2. Теоретическое введение

Статический анализ кода – процесс выявления ошибок и недочетов в исходном коде программ, который также можно рассматривать как автоматизированный процесс обзора кода. С его помощью можно предотвратить проблемы до их проникновения в состав основного программного кода и гарантировать, что новый код соответствует стандарту. Такое раннее обнаружение особенно полезно для проектов больших

встраиваемых систем, где разработчиками не могут использоваться другие средства анализа до тех пор, пока программное обеспечение не будет завершено настолько, чтобы его было можно запустить на целевой системе. Используя разные техники анализа, например, проверку абстрактного синтаксического дерева (abstract syntax tree, AST) и анализ кодовых путей, инструменты статического анализа могут выявить скрытые уязвимости, логические ошибки, дефекты реализации и другие проблемы, что может происходить как на этапе разработки на каждом рабочем месте, так и во время компоновки системы. Так, при статическом анализе могут просматриваться все потенциальные пути исполнения кода – при обычном тестировании такое происходит редко, если только в проекте не требуется обеспечения 100%-го покрытия кода. Например, при статическом анализе можно обнаружить программные ошибки, связанные с краевыми условиями или ошибками путей, не тестируемыми во время разработки. Поскольку во время статического анализа делается попытка предсказать поведение программы, основанное на модели исходного кода, то иногда обнаруживается «ошибка», которой фактически не существует – это так называемое «ложное срабатывание» (false positive). Однако, многие современные средства статического анализа реализуют улучшенную технику, чтобы избежать подобной проблемы и выполнить исключительно точный анализ. В итоге, задачи, решаемые программами статического анализа кода, можно разделить на 3 большие категории: Выявление ошибок в программах. Рекомендации по оформлению кода. Некоторые статические анализаторы позволяют проверять, соответствует ли исходный код, принятому в компании стандарту оформления кода. Имеется в виду контроль количества отступов в различных конструкциях, использование пробелов/символов табуляции и так далее. Подсчет метрик. Метрика программного обеспечения – это мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификаций. Существует большое количество разнообразных метрик, которые можно подсчитать, используя те

или иные инструменты. Анализаторы исходного кода — класс программных продуктов, созданных для выявления и предотвращения эксплуатации программных ошибок в исходных кодах. Все продукты, направленные на анализ исходного кода, можно условно разделить на три типа: Первая группа включает в себя анализаторы кода веб-приложений и средства по предотвращению эксплуатации уязвимостей веб-сайтов. Вторая группа — анализаторы встраиваемого кода, позволяющие обнаружить проблемные места в исходных текстах модулей, предназначенных для расширения функциональности корпоративных и производственных систем. К таким модулям относятся программы для линейки продуктов 1С, расширения CRM-систем, систем управления предприятием и систем SAP. Последняя группа предназначена для анализа исходного кода на различных языках программирования, не относящихся к бизнес-приложениям и веб-приложениям. Такие анализаторы предназначены для заказчиков и разработчиков программного обеспечения. В том числе данная группа анализаторов применяется для использования методологии защищенной разработки программных продуктов. Анализаторы статического кода находят проблемы и потенциально уязвимые места в исходных кодах и выдают рекомендации для их устранения. Стоит отметить, что большинство из анализаторов относятся к смешанным типам и выполняют функции по анализу широкого спектра программных продуктов — веб-приложений, встраиваемого кода и обычного программного обеспечения. Анализаторы могут содержать различные механизмы анализа, но наиболее распространенным и универсальным является статический анализ исходного кода — SAST (Static Application Security Testing), также существуют методы динамического анализа — DAST (Dynamic Application Security Testing), выполняющие проверки кода при его исполнении, и различные гибридные варианты, совмещающие разные типы анализов.

Динамический анализ является самостоятельным методом проверки, который может расширять возможности статического анализа или применяться

самостоятельно в тех случаях, когда доступ к исходным текстам отсутствует.

3. Ход работы

Часть 1

Для тестирования используется проект API, созданный для имитации работы контактной книги.

1. Проверить ранее сделанный учебный проект несколькими статическим анализатором (от 3-5). Сделать вывод о адекватности найденных ошибок.

Для выполнения используем анализатор, встроенный в IntelliJ IDEA (Рисунок 3.1), Visual Studio Code (Рисунок 3.2), Eclipse (Рисунок 3.3).

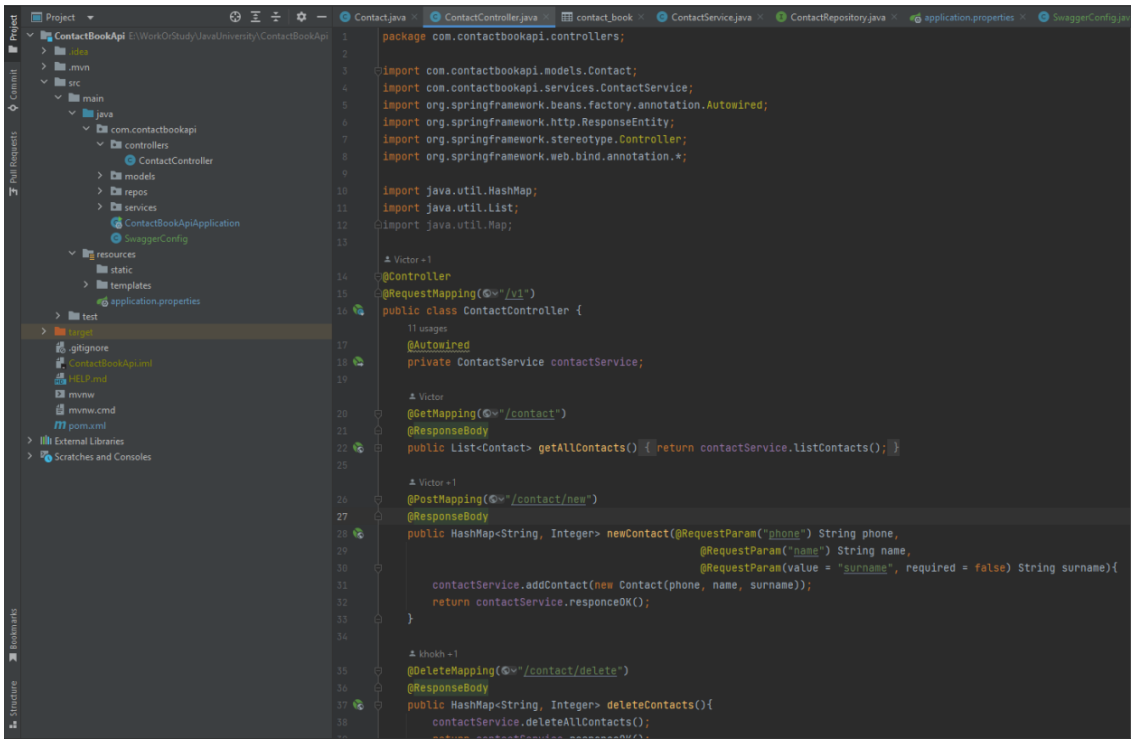

Рисунок 3.1 – IntelliJ IDEA статический анализатор

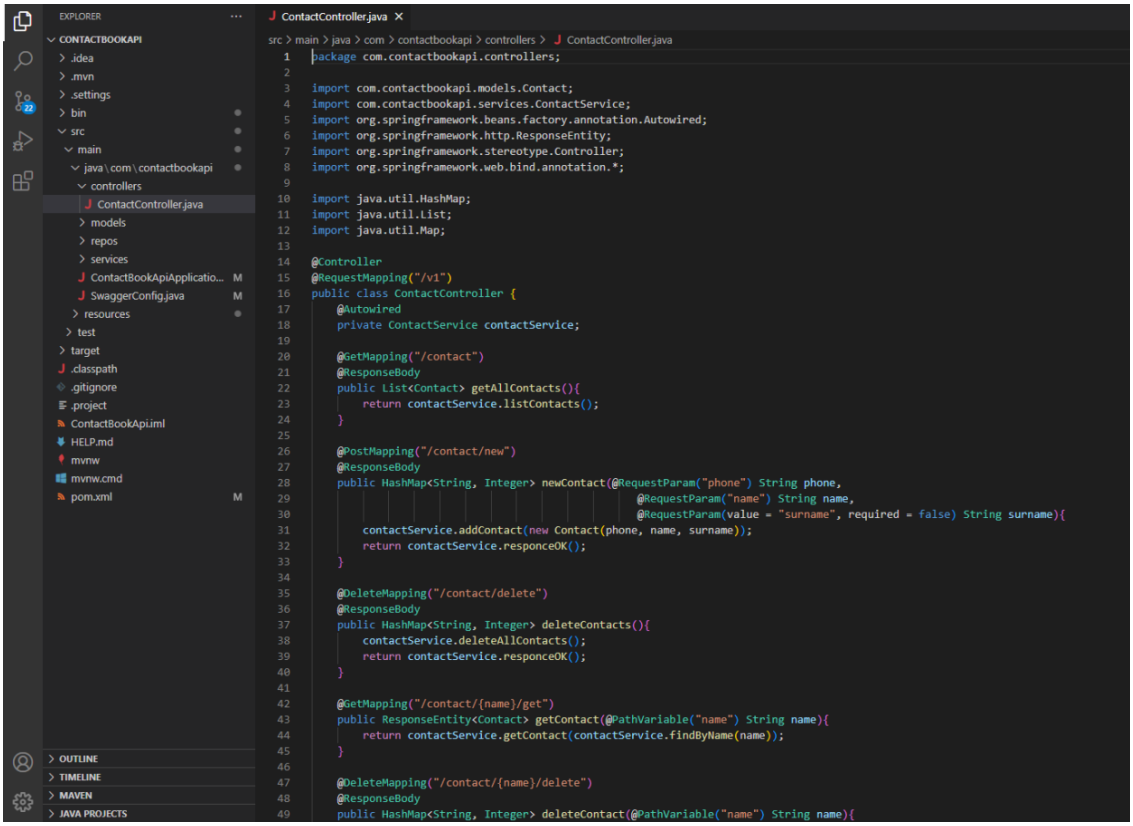

Рисунок 3.2 – Visual Studio Code статический анализатор

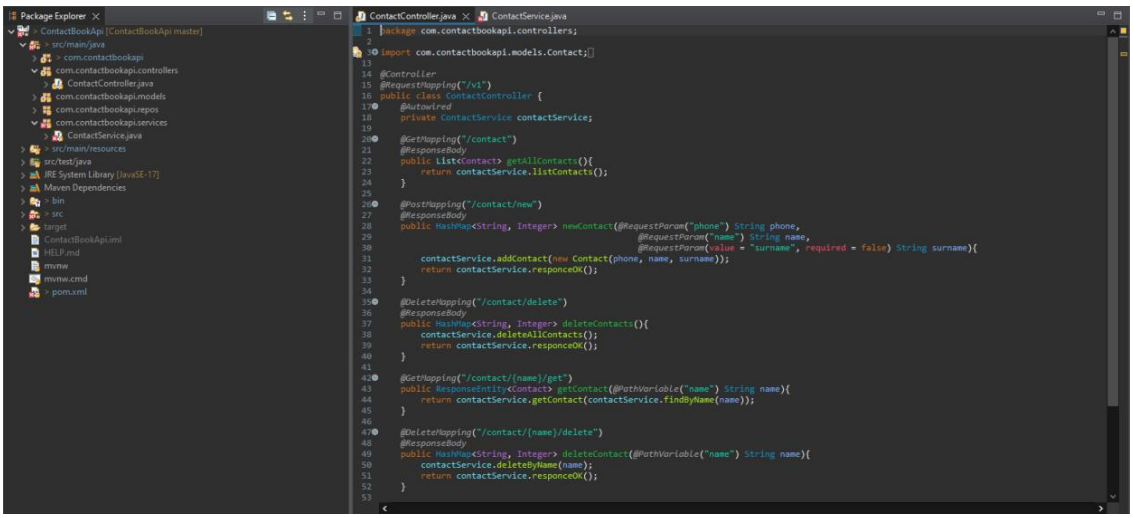

Рисунок 3.3 – Eclipse статический анализатор

2. Внести разные типы ошибок и проверить работу анализаторов.

Внесем ошибки (Рисунок 3.4 – 3.6).

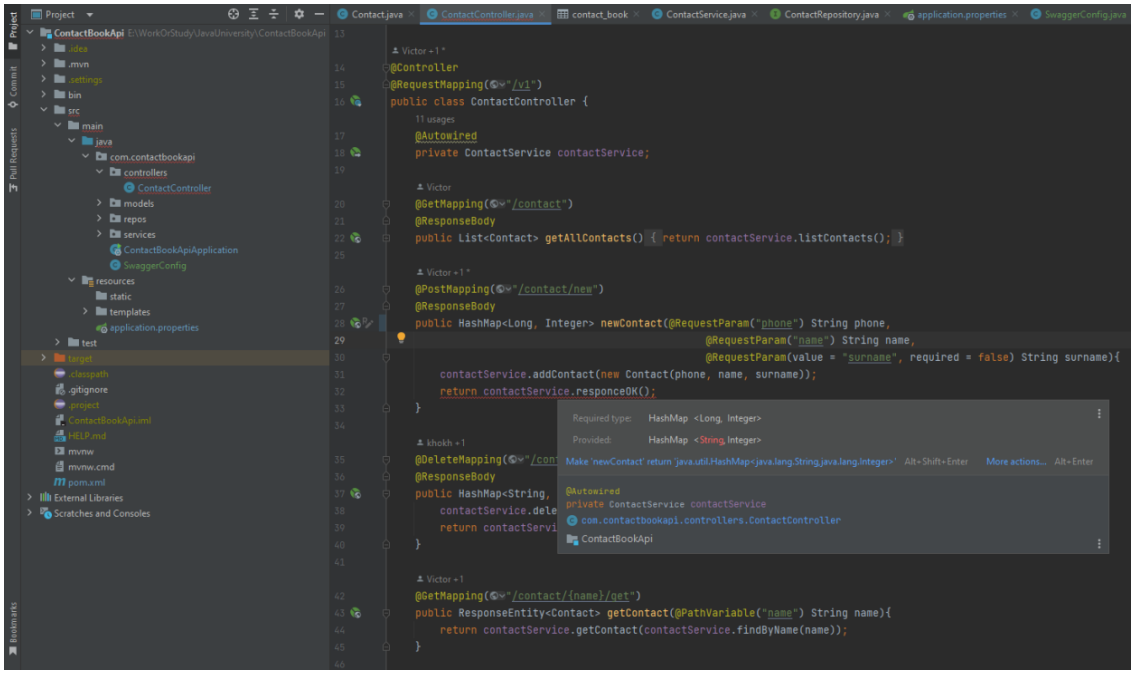

Рисунок 3.4 – Ошибка №1

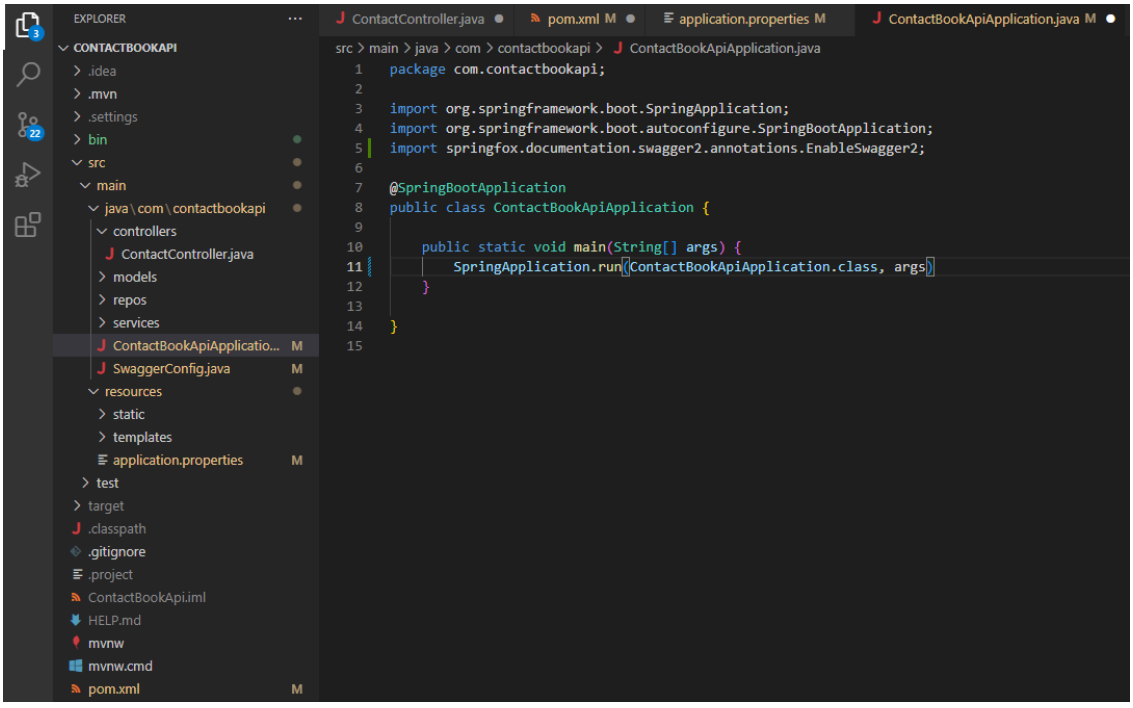

Рисунок 3.5 – Ошибка №2

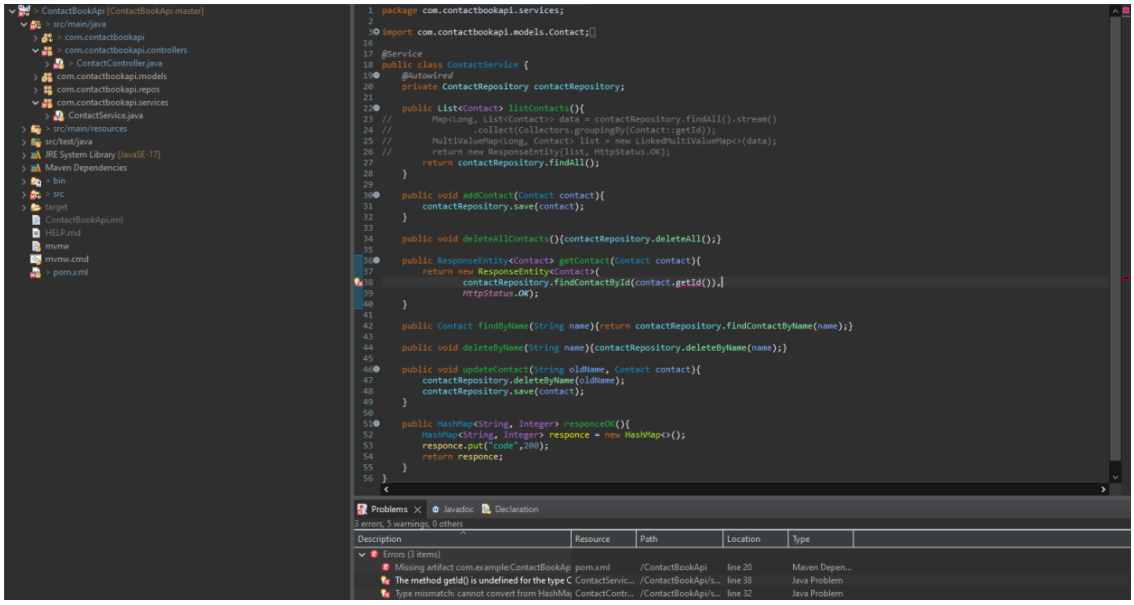

Рисунок 3.6 – Ошибка №3

3. Сделать вывод о целесообразности статического анализа.

Анализатор помогает избежать ошибки названия, отсутствия присвоения, возврат некорректного значения, однако не помогает, если переменная названа одним названием с существующей, или происходит сложная логика, например, с возвратом NULL.

4. Предложить **варианты** написания своего анализатора для решения проблем из своего опыта, которые возникали слишком часто или имели негативные последствия. Реализованы ли такие средства в текущих анализаторах, что Вы попробовали?

Большинство идей уже реализовано в существующих анализаторах, введение ИИ не может привести с хороших результатов из-за возможных костыльных решений, что будет браковаться ИИ. Так же стиль у всех разный, со своими привычками и фишками, кроме того, может быть нарушено авторское право.

5. Проанализировать учебный код динамическим анализатором.

В качестве динамического анализатора выбрано встроенное расширение «IntelliJ IDEA».

6. Внести ошибки и проверить адекватность работы динамического анализатора.

Внедрим ошибки и посмотрим результат (Рисунок 3.7).

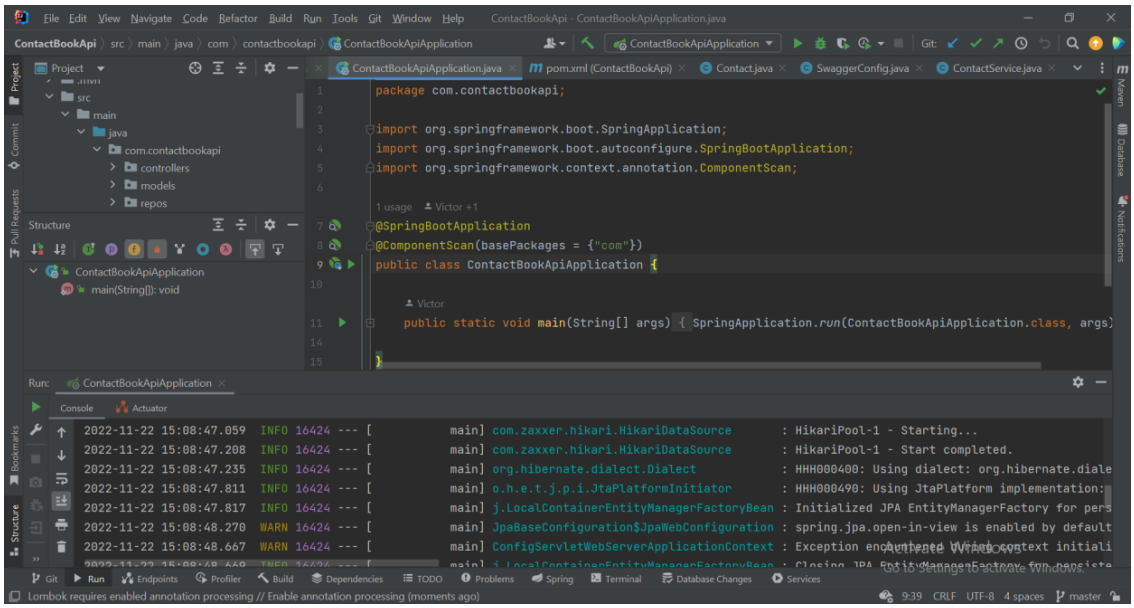

Рисунок 3.7 – Предупреждение от динамического анализатора.

7. Оценить возможность создания автоматной модели по коду пример.

Автоматная модель позволяет объективно оценить сложность будущей программы. Она зависит от числа состояний автомата, количества входных/выходных каналов, связности графа, т.е. множества и разнообразия переходов, и сложности предикатов и действий. Предикаты и действия в случае ПЛК достаточно просты. Обычно это: прочитать вход ПЛК в случае предиката и/или установить выходной сигнал в случае действия. А вот логика алгоритма, особенно при взаимодействии с другими функциональными блоками, способна доставить массу хлопот.

В данном примере кода используются присвоения, поэтому построение для выявления ошибок достаточно сложны.

Выводы

В ходе выполнения данной практической работы был получен опыт тестирования с помощью статических и динамических анализаторов кода.

Список использованных источников

1. Лекции Петренко А.А. по дисциплине «Тестирование и верификация программного обеспечения». МИРЭА, 2023.

2. Python's documentation, tutorials, and guides are constantly evolving. [Электронный ресурс] – 2023 – Режим доступа: https://www.python.org/doc/, свободный.

3. Python's documentation, tutorials, and guides are constantly evolving. [Электронный ресурс] – 2023 – Режим доступа: https://www.python.org/doc/, свободный.

4. TestComplete 15 Documentation. [Электронный ресурс] – 2023 – Режимдоступа: https://support.smartbear.com/testcomplete/docs/app-testing/desktop/index.html, свободный