**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**Факультет** ВТ **«К защите»**

**Кафедра** ВПМ Зав. кафедрой \_\_\_\_\_\_\_\_ Пылькин А.Н.

**Направление** 09.03.04 “\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2018 г.

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к выпускной квалификационной работе бакалавра на тему

**«Разработка программной системы автоматизации составления отчетов по практическим заданиям»**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
| Студент: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Николаев Н.А.) |
| Руководитель проекта: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Пруцков А.В.) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2018 г.

Аннотация

Пояснительная записка – 105 страниц, 15 таблиц, 65 рисунок, 33 источника.

Автоматизированная система составления отчетов, проверка качества кода.

Цель работы – разработка программной системы автоматизации процессов проверки качества исходного кода и составления отчетов по практическим работам.

Назначение разработанной программной системы – автоматизация процесса составления отчетов по практическим работам.

В результате разработана программная система, обладающими следующими функциями:

- Автоматическое формирование отчета по практической работе в соответствии с шаблоном;

- Использование архива с исходным кодом в качестве входных данных для формирования отчета;

- Проведение проверки стиля и качества исходного кода в соответствии с определенными правилами;

- Защита отчета от последующих изменений.

Предполагается внедрение в дисциплину «Программирование на Java».

*Аnnotation*

*Explanatory note – 105 pages, 15 tables, 65 figures, 33 sources.*

*Automated reporting system, code quality check.*

*The goal of the work is the development of a software system for automating the processes of checking the quality of the source code and drawing up reports on practical work.*

*The purpose of the developed software system is the automation of the process of drawing up reports on practical work.*

*As a result, a software system was developed that performs the following functions:*

*- Automatic generation of a report on practical work in accordance with the template;*

*- Using an archive with source code as input to the report;*

*- Checking the style and quality of the source code in accordance with certain rules;*

*- Protecting the report from subsequent changes.*

*It is supposed that it will be introduced into the discipline "Programming in Java".*

Содержание

[ВВЕДЕНИЕ 7](#_Toc516612415)

[1. Технико-экономическое обоснование темы 10](#_Toc516612416)

[2. Анализ предметной области 17](#_Toc516612417)

[3. Разработка алгоритмов работы системы 44](#_Toc516612418)

[4. Разработка программной системы 50](#_Toc516612419)

[4.1 Разработка пользовательского интерфейса 50](#_Toc516612420)

[4.2 Реализация пользовательского интерфейса 53](#_Toc516612421)

[4.3 Структура программной системы 57](#_Toc516612422)

[5. Разработка программной документации 69](#_Toc516612423)

[5.1 Руководство программиста 69](#_Toc516612424)

[5.2 Руководство пользователя 70](#_Toc516612425)

[6. Тестирование 83](#_Toc516612426)

[ЗАКЛЮЧЕНИЕ 87](#_Toc516612427)

[Список использованных источников 88](#_Toc516612428)

[Приложение. Листинг наиболее значимых частей программы 91](#_Toc516612429)

# ВВЕДЕНИЕ

В настоящий момент в высших учебных заведениях на технических специальностях всегда преподаются дисциплины связанные с объектно-ориентированным программированием. И при выполнении практических заданий при сдаче требуется составлять отчет. Эти отчеты имеют одинаковый шаблон и студенту остается только заполнить необходимые разделы. Также при необходимости вставки различных участков кода процесс формирование отчета может занимать длительное время. Преподаватель тратит больше времени на правильность формирования отчета, так как этот процесс создания отчета выполняется вручную и есть шанс, что студент допустит ошибку. Этот процесс можно автоматизировать. Также при автоматизации можно добавить дополнительно проверку качества исходного кода, написанного при выполнении практического задания.

Целью данной выпускной квалификационной работы является разработка программной системы автоматизации процессов проверки качества исходного кода и составления отчетов по практическим работам. В качестве исходных данных данная система использует архив с проектом, по которому необходимо составлять отчет и шаблон для отчета. Данную программную систему планируется использовать в дисциплине «Программирование на *Java*».

В разделе «Технико-экономическое обоснование темы» рассматривается текущее состояние процесса формирования отчетов по практическим работам. Так же дается характеристика тех этапов, которые можно автоматизировать. Выявляется актуальность данной темы, и предлагаются варианты использования программной системы. Происходит сравнение с аналогами, существующими на данный момент. Разбирается возможность создание программной системы с использованием существующих технологий, так же предлагаются стратегии для привлечения денежных ресурсов. В конце идет обоснование выбранных средств разработки.

В разделе «Анализ предметной области» происходит построение вариантов использования программной системы. Детально описываются все прецеденты, их назначение, действующие лица, предусловия и постусловия, основные и альтернативные поток, приоритет и частота использования. Для каждого прецедента с помощью диаграмм деятельности показаны действия, которые специфицируют его поведение.

В разделе «Разработка алгоритмов работы системы» строится базовая архитектура программной системы. Выделяются основные модули, и описывается их назначение. Так же строится базовый алгоритм работы всей программной системы. И после этого детализируются некоторые основные этапы: проверка качества кода, формирование отчета, защита отчета от изменений.

В разделе «Разработка программной системы» происходит проектирование и разработка пользовательского интерфейса. В разделе показаны макеты и реализация всех окон программной системы. Так же строятся физическая структура программной системы. С помощью диаграмм классов показано, из каких компонентов состоит реализация программной системы, и какие шаблоны были использованы в процессе разработки, и с помощью диаграмм размещения представлено как данная система будет развернута.

В разделе «Разработка программной документации» представлена документация для программиста и пользователя. Документация для программиста дает представлению новому разработчику, каким образом можно изменять токующий функционал системы или добавлять новый. Документация пользователя помогает пользователю в процессе обучения работы с системой. Она показывает все возможности системы, доступные для пользователя.

В разделе «Тестирование» проводится тестирование функционала системы с использованием тест-кейсов. Тестирование направлено на проверку работоспособности основного функционала. В ходе тестирования были выявлены и устранены найденные ошибки в работе системы.

В ходе разработки была использована интегрированная среда разработки программного обеспечения *IntelliJ IDEA* [30]. Она сильно ускоряет процесс разработки приложений на *Java* [Романчик3]. Так как включается в себя множество плагинов направленных на помощь программисту, статический анализ кода, подсказки по использованию методов и функций, автогенерация кода. Данное средство для студентов распространяется бесплатно.

В результате была разработана программная система автоматизации процессов проверки качества исходного кода и составления отчетов по практическим работам.

# Технико-экономическое обоснование темы

В процессе обучения в вузе студентам при выполнении практических работ требуется составлять отчеты. Процесс формирования отчета и его проверки занимает много времени и сил. Этот процесс можно проиллюстрировать с помощью диаграммы потоков данных [14, с. 118-119]. Построив ее, мы сможем выделить этапы, которые можно автоматизировать. На рисунке 1 представлена диаграмма потоков данных при формировании отчетов по дисциплине «Программирование на *Java*».

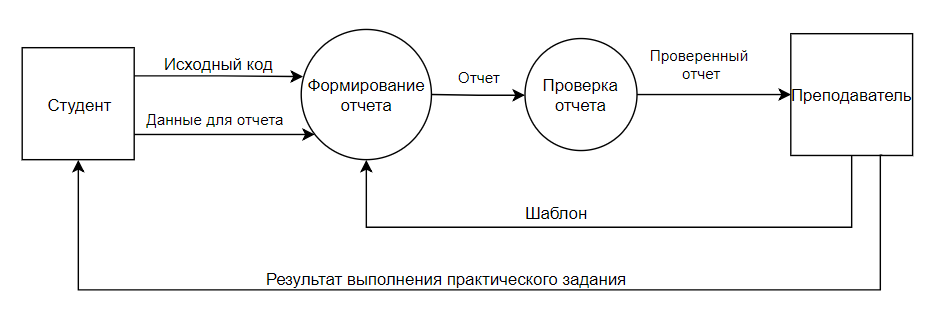


Рисунок 1 – Диаграмма потоков данных при формировании отчета

Студент выполняет практическое задание. И переходить к формированию отчета. Он берет шаблон, которые он получил от преподавателя, и заполняет необходимые разделы. В процессе формирования отчета студент вставляет в отчет наиболее значимые участки исходного кода, который был написан в процессе выполнения им задания. Вставляет в отчет данные о себе, такие как: ФИО, группа, номер практического задания. И сохраняет получившийся документ. Далее этот отчет отправляет преподавателю на проверку. При проверке преподаватель вручную проверяет отчет на предмет использования верного шаблона и соответствия документа предъявленным требованиям. Все выявленные ошибки и недостатки высылаются студентам, и после исправления студент присылает практическую работу еще раз. Так как разделы отчета, которые необходимо заполнить, для разных практических заданий остаются одинаковые, меняется только их содержимое, то этап формирования отчета можно автоматизировать. Так же на этом этапе можно проводить проверку качества написания исходного кода [11, 10] и сообщать студенту о найденных недостатках, чтобы он мог их сразу устранить, без участия преподавателя. Результаты проверки вставляются в отчет, и преподаватель при изучении отчета сможет видеть эти результаты и ему не потребуется запускать вручную статический анализатор кода при проверке исходного кода выполненного практического задания. Это помогает сократить время на принятие решения о результатах выполнения практического задания. Так же у преподавателя есть возможность обратить больше времени на архитектуру приложения.

Шаблон, по которому составляется отчет, меняется время от времени. И при изменении его приходится рассылать заново всем студентам. Студент может забыть об этом изменении и использовать старый шаблон, и эти действия могут увеличить время сдачи практического задания. При использовании программной системы шаблон можно загружать в автоматическом режиме из интернета. В таком случае преподавателю будет необходимо обновить шаблон в хранилище, и все студенты увидят это изменение. Это сильно облегчит возможность обновления шаблоны отчета. И уменьшит количество допускаемых студентом ошибок. Но также необходимо оставить возможность указания файла с шаблоном отчетом локально на компьютере, в случае если необходимо использовать свой шаблон или нет доступа к интернету.

Чтобы преподаватель был уверен, что отчет создан именно данной программной системой и содержит полную информацию о произведенной проверке исходного кода без изменений, выполненных студентом, полученный отчет защищается от изменений. Это убирает возможность его редактирования после формирования.

Целью данной выпускной квалификационной работы является разработка программной системы автоматизации процессов проверки качества исходного кода и составления отчетов по практическим работам, которая имеет следующие функции:

* Автоматическое формирование отчета по практической работе в соответствии с шаблоном;
* Использование архива с исходным кодом в качестве входных данных для формирования отчета;
* Проведение проверки стиля и качества исходного кода в соответствии с определенными правилами;
* Защита отчета от последующих изменений.

Для достижения цели необходимо выполнить следующие задачи:

* Создать модуль для формирования отчетов по заданным требованиям;
* Создать модуль автоматической проверки качества написания кода;
* Создать модуль защиты сформированного отчета от изменений.

Для решения задачи формирования автоматического отчетов [16] в настоящее время в большинстве языков программирования есть возможность работы с файлами, изменять их содержимое, сохранять и удалять их.

Для того чтобы, автоматически проверить качество написания исходного кода, необходимо выделить критерии по которым это можно оценить [15]. Главным критерием при оценке качества кода является его работоспособность. Но, кроме того, чтобы код просто работал, он должен отвечать дополнительно некоторым требованиям, и в первую очередь это читаемость и управляемость. Под читаемостью понимается возможность в короткие сроки понять алгоритм, который был реализован в данном участке кода, и оценить, как будет вести себя программа в том или ином случае. При автоматической проверке этого критерия можно использовать цикломатическую сложность кода [6, с. 92-93]. И в зависимости от полученного значения выдавать результат. Ведь чем больше ветвлений в коде, тем сложнее его понять. Еще один способ увеличить читаемость кода - это использовать соглашение по написанию кода, если студенты будут придерживаться одних и тех же правил написания кода, то преподавателю будет гораздо легче проверять написанный код. Ему не надо будет тратить время, чтобы привыкнуть к стилю написания каждого студента. Но добиться того, чтобы каждый студент следовал этим правилам достаточно трудно. При рассмотрении языка программирования *Java* можно использовать статический анализатор кода *Checkstyle* [33, с. 219-225], это должно сильно упростить решаемую задачу. Управляемость кода - это возможность в минимальные сроки внести требуемые изменения в код, при этом избежав сильного изменения уже написанного кода. Этому также соответствует один из принципов *SOLID* [28]. А именно принцип открытости/закрытости (*open-closed*), который говорит нам о том, что программные сущности должны быть открыты для расширения, но закрыты для модификации. И если мы хотим реализовать какую-то дополнительную функциональность, то нам необходимо только расширять уже написанные классы. Этот критерий оценить сложнее, но можно использовать такие средства как *PMD* [33, с. 219-225] и *FindBugs* [33, с. 219-225]. В них заложено множество шаблонов ошибок и некорректного кода. Так же, *FindBugs* анализирует байт-код программы, что позволяет найти больше уязвимостей в исходном коде, по сравнению с анализом не скомпилированного кода.

Для защиты от изменений распространен метод получения контрольной суммы файла [8, с. 593-598]. По этой сумме можно проверить подлинность файла. Наиболее популярные алгоритмы вычисления контрольной суммы: *SHA256, MD5, SHA1* [21]. Для того чтобы обеспечить более точный результат, мы будем использовать в исходных данных для формирования отчета не только содержимое отчета, но и содержимое шаблона. Чтобы преподаватель мог быть уверен, что отчет составлен по верному шаблону. После того как отчет получен, в метаинформацию необходимо записать полученную контрольную сумму, по которой преподаватель будет знать изменялся ли отчет. Эти возможности есть в современных языках программирования.

В результате написания программной системы автоматической проверки качества исходного кода и формирования отчетов по практическим заданиям диаграмма потоков данных выглядит так, как представлено на рисунке 2.

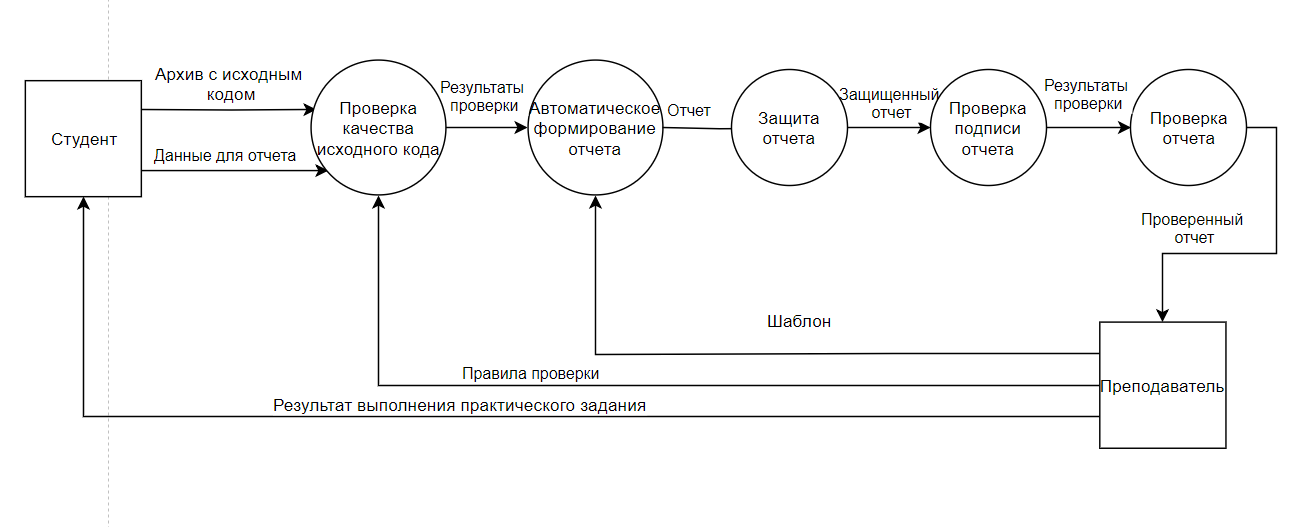


Рисунок 2 – Диаграмма потоков данных при формировании отчета с помощью программной системы

При формировании с помощью программной системы студенту необходимо будет предоставить данные для отчета в виде архива с исходным кодом и дополнительной информации о себе, которая будет указана в шаблоне. Система проверяет исходный код программы с помощью статических анализаторов [2]. Далее сформирует отчет и защитит его от изменений. У преподавателя должна быть программа для проверки подписи полученных отчетов, чтобы он был уверен, что в данный отчет не вносились изменения, после того, как студент его сформировал с помощью программной системы автоматического формирования отчетов. После этого преподавателю останется проверить отчет на предмет ошибок и недочетов в содержании и исходном коде.

Для привлечения денежных ресурсов можно использовать различные стратегии стратегий. При размещении программной системы на веб ресурсе можно использовать контекстную рекламу [12]. Минимальная ставка за показ объявления в верхней части страницы составляет 1,24 рубля [27]. Максимальная ставка 8.09 рубля. В качестве привлечения дополнительной аудитории можно использовать модуль формирования отчета в качестве плагина для уже существующих решений, например для *SonarQube* [26]. Так же данная программная система может быть добавлена в состав учебного модуля, который формируется для продажи другому высшему учебному заведению.

Для частичного решения поставленных задач, которая решает данная программная система, в настоящее время существуют различные системы. Рассмотрим в сравнении разработанной программной системой некоторые из них *SonarQube* и *Code Climate* [25].

*SonarQube* - является бесплатным проектом с открытым исходным кодом для непрерывного анализа и измерения качества кода, но некоторый функционал может требовать платного доступа. Имеет следующие возможности: поддержка многих языков программирования *Java, JavaScript, Objective-C, C#, C, C++, , Swift, PHP, Python* и др.; предоставление отчетов о соблюдении стандартов написания кода; считает процент покрытия кода модульными тестами; сохраняет историю метрик; интегрируется с *Maven, Ant, Grade* и другими распространенными системами непрерывной интеграции [24, с.135 - 151]. Для расширения функционала есть возможность установки дополнительных плагинов. Так же удобно представлен просмотр статистики по проектам. Для использования требуется наличие сервера. Для системы, которая будет использоваться в качестве решения поставленной задачи, имеет избыточный функционал, в котором нет необходимости, так же для использования требуется много ресурсов компьютера.

*Сode Climate* – это средство с открытым исходным кодом для автоматического обзора кода для *GitHub и GitLab.* Данное средство включает в себя статические анализаторы кода для многих языков программирования и выдает отчет о результатах проверки. Использование данного средства в образовательном процессе является затруднительным из-за того, что используется система контроля версий *git* [22]. Для обучения требуется затратить больше времени. Недостатком всех рассмотренных систем является то, что они не поддерживают автоматическое составление отчетов в соответствии с шаблоном и не позволяют защитить сформированный отчет. И если в случае с *SonarQube* мы можем написать дополнительные плагины, которые будут решать данные проблемы. То при использовании *Code Climate* это будет сделать достаточно проблематично. Так как нет готовой системы модификации данного программного средства. Так же оба этих средства направлены на коммерческую разработку программного обеспечения, а не для обучения.

В собственной программной системе отсутствуют рассмотренные недостатки, реализован только необходимый функционал, решение не требует много ресурсов компьютера и при необходимости увеличить функционал модули формирования и защиты отчета можно использовать в качестве плагина для *Sonar Qube*.

В качестве средства разработки был выбран язык программирования *Java*.Так как программы, написанные на *Java*, исполняются с использованием *JVM (Java virtual machine)* [17, с. 452-461]. Данная виртуальная машина исполняет байт-код, предварительно создающийся компилятором *Java* или другого языка, который может компилироваться в *Java* байт-код. Это позволяет писать кросс-платфоменные приложения [23], которые можно запускать на различных системах, на которых установлена *jvm*. Так же в созданный jar файл есть возможность внедрить мини jvm и получить на выходе исполняемый файл, это позволит запускать приложение даже если на компьютере не установлена виртуальная машина. В *JVM* поддерживает множество языков, поэтому выбор конкретного языка зависел от возможности создания удобного пользовательского интерфейса, способами работы с файлами и доступностью внедрения статических анализаторов кода. В случае с *Java* для пользовательского интерфейса используется *JavaFX* [7], это средство хорошо подходит для легкого и быстрого создания десктопных приложений.

# Анализ предметной области

Для формирования требований используются диаграммы прецедентов. Они позволяют дать полную картину всех возможностей моделируемой программной системы, видя которые пользователь получает конкретный, измеримый и нужный ему результат.

Диаграмма прецедентов показывает отношения между прецедентами и актерами. Она помогает описать систему на концептуальном уровне и является частью модели прецедентов [5].

Под прецедентом понимается функция системы, которой может воспользоваться пользователь, чтобы получить необходимый ему результат. Прецедент описывает вариант использования системы для пользователя. Обычно это типовые варианты использования. Диаграмма вариантов использования часто применяется для составления внешних требований для системы.

На диаграмме вариантов использования могут быть показаны несколько отношений: обобщение прецедента, включение прецедента (*«include»*) и расширение прецедента (*«extend»*).

На рисунке 3 видны все возможности, которыми обладают пользователи программной системы автоматизации системы отчетов по практическим заданиям.

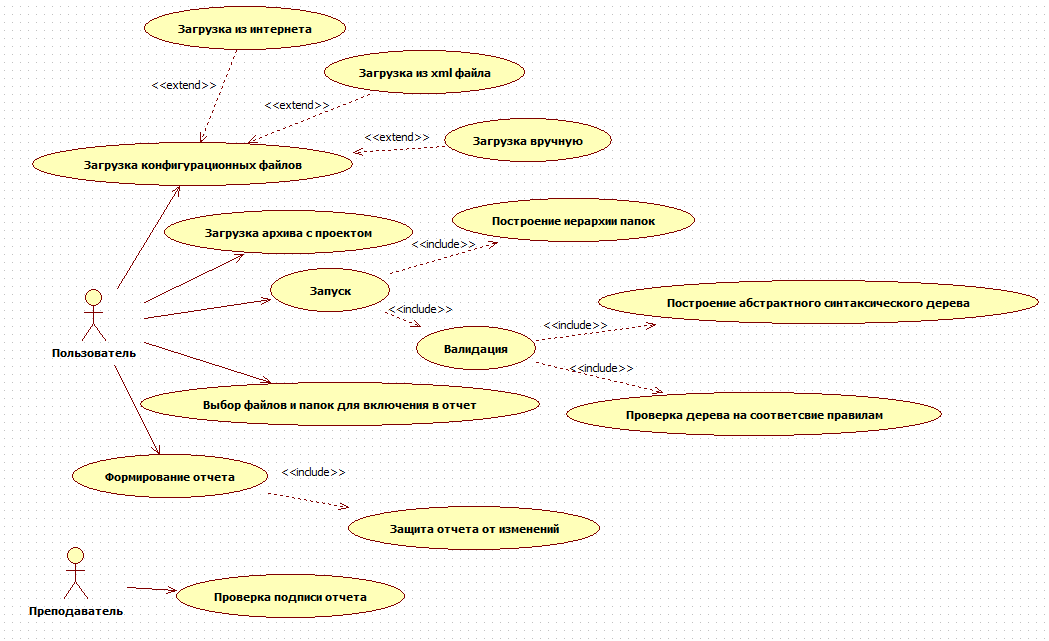


Рисунок 3 – Диаграмма прецедентов программной системы формирования отчетов

**Спецификация диаграммы прецедентов**

В таблице 1 представлено описание прецедента «Загрузка конфигурационных файлов»

Таблица 1 – Загрузка конфигурационных файлов

|  |  |
| --- | --- |
| **Характеристика** | **Описание характеристики** |
| Действующие лица | Пользователь, система |
| Краткое описание | Представляет собой загрузку конфигурационных файлов, необходимых для работы программы |
| Предусловия | Пользователем было принято загрузить файлы с настройками в систему |
| Постусловия | Файлы с настройками загружены в систему |
| Основной поток (нормальное течение) | 1. Пользователь нажимает кнопку “Загрузить шаблон” 2. Выполняется прецедент «Загрузка из интернета» |
| Альтернативный поток (альтернативные течения) | 1а. [Файлы настроек ранее уже загружались, пользователь не хочет их менять]  1а1. Выполняется прецедент «Загрузка из xml файла»  1а2. Выполнение прецедента завершается  1б. [Пользователь решил загрузить файлы настроек вручную]  1б1. Выполняется прецедент «Загрузка файлов вручную»  1б2. Выполнение прецедента завершается |
| Приоритет (Критично | Важно | Желательно) | Критично |
| Частота использования (Всегда | Часто | Иногда | Редко | Один раз) | Всегда |

В таблице 2 представлено описание прецедента «Загрузка из xml файла»

Таблица 2 – Загрузка из xml файла

|  |  |
| --- | --- |
| **Характеристика** | **Описание характеристики** |
| Действующие лица | Cистема |
| Краткое описание | Производится загрузка конфигурационных файлов, информация о которых содержится в xml файле |
| Предусловия | Система начала свою работу |
| Постусловия | Система загрузила конфигурационные файлы |
| Основной поток (нормальное течение) | Система получает путь до конфигурационных файлов из xml  Система загружает конфигурационные файлы, используя полученные пути |
| Альтернативный поток (альтернативные течения) | 1а. [xml файла не существует]  1a1. Система выводит сообщение об ошибке  1a2. Выполнение прецедента завершается  2а. [конфигурационные файлы не найдены]  2a1. Система выводит сообщение об ошибке  2a2. Выполнение прецедента завершается |
| Приоритет (Критично | Важно | Желательно) | Критично |
| Частота использования (Всегда | Часто | Иногда | Редко | Один раз) | Часто |

В таблице 3 представлено описание прецедента «Загрузка из интернета»

Таблица 3 – Загрузка из интернета

|  |  |
| --- | --- |
| **Характеристика** | **Описание характеристики** |
| Действующие лица | Пользователь, система |
| Краткое описание | Производится загрузка конфигурационных файлов из интернета, по указанному *url* |
| Предусловия | Система начала свою работу |
| Постусловия | Система загрузила конфигурационные файлы |
| Основной поток (нормальное течение) | Система запрашивает *url* путь до конфигурационных файлов  Система загружает конфигурационные файлы, используя полученный путь |
| Альтернативный поток (альтернативные течения) | 2а. [*url* не существует]  2a1. Система выводит сообщение об ошибке  2a2. Выполнение прецедента завершается  2б. [конфигурационные файлы по указанному *url* не найдены]  2б1. Система выводит сообщение об ошибке  2б2. Выполнение прецедента завершается  2в. [отсутствует подключение к интернету]  2в1. Система выводит сообщение об ошибке  2в2. Выполнение прецедента завершается |
| Приоритет (Критично | Важно | Желательно) | Критично |
| Частота использования (Всегда | Часто | Иногда | Редко | Один раз) | Часто |

В таблице 4 представлено описание прецедента «Загрузка файлов вручную»

Таблица 4 – Загрузка файлов вручную

|  |  |
| --- | --- |
| **Характеристика** | **Описание характеристики** |
| Действующие лица | Пользователь, система |
| Краткое описание | Производится загрузка конфигурационных файлов вручную |
| Предусловия | Пользователем было принято загрузить файлы с настройками в систему вручную. |
| Постусловия | Система загрузила конфигурационные файлы |
| Основной поток (нормальное течение) | Пользователь нажимает кнопку “Загрузить шаблон”  Пользователь выбирает файл с шаблоном.  Пользователь нажимает кнопку «Выбрать»  Пользователь нажимает кнопку “Загрузить конфигурацию *checkstyle*”  Система открывает форму для загрузки файлов  Пользователь выбирает файл с конфигурацией *checkstyle*.  Пользователь нажимает кнопку «Выбрать» |
| Альтернативный поток (альтернативные течения) | 3a. [Пользователь решил отменить загрузку]  3а1. Пользователь нажимает кнопку «Отмена».  3а2. Выполнение прецедента завершается.  6a. [Пользователь решил отменить загрузку]  6а1. Пользователь нажимает кнопку «Отмена».  6а2. Выполнение прецедента завершается. |
| Приоритет (Критично | Важно | Желательно) | Критично |
| Частота использования (Всегда | Часто | Иногда | Редко | Один раз) | Часто |

В таблице 5 представлено описание прецедента «Загрузка архива с проектом»

Таблица 5 – Загрузка архива с проектом

|  |  |
| --- | --- |
| **Характеристика** | **Описание характеристики** |
| Действующие лица | Пользователь, система |
| Краткое описание | Производится загрузка архива с проектом |
| Предусловия | Пользователем было принято решение загрузить архив с проектом |
| Постусловия | Система загрузила архив с проектом |
| Основной поток (нормальное течение) | 1. Пользователь нажимает кнопку “Выбрать архив с проектом” 2. Система открывает форму для загрузки файлов 3. Пользователь выбирает архив 4. Пользователь нажимает кнопку «Выбрать» |
| Альтернативный поток (альтернативные течения) | 3a. [Пользователь решил отменить загрузку]  3а1. Пользователь нажимает кнопку «Отмена».  3а2. Выполнение прецедента завершается. |
| Приоритет (Критично | Важно | Желательно) | Критично |
| Частота использования (Всегда | Часто | Иногда | Редко | Один раз) | Часто |

В таблице 6 представлено описание прецедента «Запуск»

Таблица 6 – Запуск

|  |  |
| --- | --- |
| **Характеристика** | **Описание характеристики** |
| Действующие лица | Пользователь, система |
| Краткое описание | Производится запуск работы системы |
| Предусловия | Загружены все необходимые файлы |
| Постусловия | Система сформировала отчет внутри себя |
| Основной поток (нормальное течение) | 1. Пользователь нажимает кнопку «Запуск» 2. Выполняется прецедент «Построение иерархии папок» 3. Выполняется прецедент «Валидация» 4. Система выводит сообщения об ошибках |
| Альтернативный поток (альтернативные течения) |  |
| Приоритет (Критично | Важно | Желательно) | Критично |
| Частота использования (Всегда | Часто | Иногда | Редко | Один раз) | Часто |

В таблице 7 представлено описание прецедента «Построение иерархии папок»

Таблица 7 – Построение иерархии папок

|  |  |
| --- | --- |
| **Характеристика** | **Описание характеристики** |
| Действующие лица | Система |
| Краткое описание | Происходит построение иерархии папок |
| Предусловия | Был загружен архив с исходными файлами и запущена работа системы |
| Постусловия | Система сформировала иерархию папок |
| Основной поток (нормальное течение) | 1. Система разархивирует архив с проектом во временную папку 2. Система строит из папок и файлов иерархию 3. Система загружает графический модуль для отображения 4. Система формирует визуальное отображение полученной иерархии 5. Система предоставляет результаты пользователю |
| Альтернативный поток (альтернативные течения) | 1a. [Архив с проектом поврежден]  1а1. Система выводит сообщение об ошибке  1а2. Выполнение прецедента завершается |
| Приоритет (Критично | Важно | Желательно) | Критично |
| Частота использования (Всегда | Часто | Иногда | Редко | Один раз) | Часто |

В таблице 8 представлено описание прецедента «Выбор файлов и папок для включения в отчет»

Таблица 8 – Выбор файлов и папок для включения в отчет

|  |  |
| --- | --- |
| **Характеристика** | **Описание характеристики** |
| Действующие лица | Пользователь, система |
| Краткое описание | Происходит выбор папок и файлов, которые будут вставлены в отчет |
| Предусловия | Сгенерированная иерархия папок |
| Постусловия | Система сохранила результаты выбора папок и файлов пользователем |
| Основной поток (нормальное течение) | Пользователь видит иерархию папок и файлов  Пользователь выбирает, какие файлы он хочет включить в отчет  Пользователь ставит галочки напротив необходимых файлов и папок |
| Альтернативный поток (альтернативные течения) |  |
| Приоритет (Критично | Важно | Желательно) | Важно |
| Частота использования (Всегда | Часто | Иногда | Редко | Один раз) | Часто |

В таблице 9 представлено описание прецедента «Валидация»

Таблица 9 – Валидация

|  |  |
| --- | --- |
| **Характеристика** | **Описание характеристики** |
| Действующие лица | Система |
| Краткое описание | Происходит валидация исходных файлов проекта на соответствие соглашению по написанию кода |
| Предусловия | Были загружены все конфигурационные файлы и архив с исходными файлами и запущена работа системы |
| Постусловия | Система выдала сообщение о результатах валидации |
| Основной поток (нормальное течение) | 1. Из архива с исходными файлами проекта по очереди получаем исходный код каждого файла 2. Из конфигурационного файла *checkstyle* получаются все проверки, которые необходимо провести 3. Выполняется прецедент «Построение абстрактного синтаксического дерева» 4. Выполняется прецедент «Проверка дерева на соответствие правилам» 5. Система выводит сообщения об ошибках |
| Альтернативный поток (альтернативные течения) | 1a. [Архив с проектом не открылся]  1а1. Система выводит сообщение об ошибке  1а2. Выполнение прецедента завершается  2a. [Конфигурационный файл *checkstyle* не открылся]  2а1. Система выводит сообщение об ошибке  2а2. Выполнение прецедента завершается |
| Приоритет (Критично | Важно | Желательно) | Критично |
| Частота использования (Всегда | Часто | Иногда | Редко | Один раз) | Часто |

В таблице 10 представлено описание прецедента «Построение абстрактного синтаксического дерева»

Таблица 10 – Построение абстрактного синтаксического дерева

|  |  |
| --- | --- |
| **Характеристика** | **Описание характеристики** |
| Действующие лица | Система |
| Краткое описание | Происходит построение абстрактного синтаксического дерева |
| Предусловия | Запущена валидация архива с исходными файлами |
| Постусловия | Система построила абстрактное синтаксическое дерево |
| Основной поток (нормальное течение) | 1. Исходный код файла разбивается на синтаксические различные структуры 2. Происходит парсинг структур 3. В зависимости от уровня структуры создаются различные уровни дерева 4. Все ветки собираются в одно дерево |
| Альтернативный поток (альтернативные течения) | 1a. [Файл не найден]  1а1. Система выводит сообщение об ошибке  1а2. Выполнение прецедента завершается |
| Приоритет (Критично | Важно | Желательно) | Критично |
| Частота использования (Всегда | Часто | Иногда | Редко | Один раз) | Часто |

В таблице 11 представлено описание прецедента «Проверка дерева на соответствие правилам»

Таблица 11 – Проверка дерева на соответствие правилам

|  |  |
| --- | --- |
| **Характеристика** | **Описание характеристики** |
| Действующие лица | Система |
| Краткое описание | Происходит проверка абстрактного синтаксического дерева на соответствие правилам |
| Предусловия | Построено абстрактное синтаксическое дерево |
| Постусловия | Система выдала отчет о результатах проверки абстрактного синтаксического дерева |
| Основной поток (нормальное течение) | 1. Получаем список проверок 2. Получаем следующий уровень абстрактного синтаксического дерева 3. Текущий уровень дерева проверяем на необходимость проверки 4. Проводит проверку 5. Выдает результаты проверки и сообщает о найденных ошибках 6. Выполнение прецедента завершается |
| Альтернативный поток (альтернативные течения) | 1a. [Файл с проверками найден]  1а1. Система выводит сообщение об ошибке  1а2. Выполнение прецедента завершается  3a. [Нет подходящей проверки]  3а1. Переходим к пункту 2  6а. [Дерево пройдено не полностью]  6а1. Переходим к пункту 2 |
| Приоритет (Критично | Важно | Желательно) | Критично |
| Частота использования (Всегда | Часто | Иногда | Редко | Один раз) | Часто |

В таблице 12 представлено описание прецедента «Формирование отчета»

Таблица 12 – Формирование отчета

|  |  |
| --- | --- |
| **Характеристика** | **Описание характеристики** |
| Действующие лица | Пользователь, система |
| Краткое описание | Происходит формирование отчета и сохранение его в файл |
| Предусловия | Выполнена валидация и выбраны файлы и папки для вставки в отчет |
| Постусловия | Система сохранила сформированный отчет в файл |
| Основной поток (нормальное течение) | 1. Пользователь нажимает кнопку «Сформировать отчет» 2. Система запрашивает у пользователя дополнительную информацию о нем 3. Пользователь заполняет все поля ввода 4. Пользователь нажимает кнопку «Ок» 5. Система вставляет в отчет введенную пользователем информацию, данные о результатах валидации, исходный код выбранных файлов. 6. Отчет сохраняется в файл |
| Альтернативный поток (альтернативные течения) | 4a. [Заполнены не все поля]  4а1. Система выводит сообщение о незаполненных полях  4а2. Выполняется переход к шагу 4  4б. [Пользователь решает отменить формирование отчета]  4б1. Пользователь нажимает кнопку «Отмена».  4б2. Выполнение прецедента завершается. |
| Приоритет (Критично | Важно | Желательно) | Важно |
| Частота использования (Всегда | Часто | Иногда | Редко | Один раз) | Часто |

В таблице 13 представлено описание прецедента «Защита отчета от изменений»

Таблица 13 – Защита отчета от изменений

|  |  |
| --- | --- |
| **Характеристика** | **Описание характеристики** |
| Действующие лица | Система |
| Краткое описание | Происходит защита отчета от изменений |
| Предусловия | Сформирован отчет |
| Постусловия | Отчет защищен от изменений |
| Основной поток (нормальное течение) | 1. Из файла отчета получается хэш-сумма 2. Из файла конфигурации *checkstyle*, шаблона и архива с проектом вычисляется хэш-сумма. 3. Полученные данные записываются в мета информацию файла с отчетом. |
| Альтернативный поток (альтернативные течения) |  |
| Приоритет (Критично | Важно | Желательно) | Важно |
| Частота использования (Всегда | Часто | Иногда | Редко | Один раз) | Часто |

В таблице 14 представлено описание прецедента «Проверка подписи отчета»

Таблица 14 – Проверка подписи отчета

|  |  |
| --- | --- |
| **Характеристика** | **Описание характеристики** |
| Действующие лица | Преподаватель, система |
| Краткое описание | Происходит проверка отчета на подлинность |
| Предусловия | Преподаватель получил отчет и архив с проектом |
| Постусловия | Отчет защищен от изменений |
| Основной поток (нормальное течение) | 1. Из файла отчета получается хэш-сумма 2. Из файла конфигурации *checkstyle*, шаблона и архива с проектом вычисляется хэш-сумма. 3. Полученные данные сравниваются с данными из мета информации файла с отчетом. |
| Альтернативный поток (альтернативные течения) |  |
| Приоритет (Критично | Важно | Желательно) | Важно |
| Частота использования (Всегда | Часто | Иногда | Редко | Один раз) | Часто |

Для того чтобы понять какие шаги необходимо предпринять для достижения цели по каждому прецеденту строятся диаграмма деятельности. На этой диаграмме показаны действия, то есть эти диаграммы являются спецификацией исполняемого поведения. Выполнение каждого элемента может быть как последовательным, так и параллельным.

**Загрузка конфигурационных файлов**

Перед началом работы программы необходимо загрузить конфигурационные файлы, для полноценной работы системы. Эти файлы представляют собой шаблон и настройки для проверки кода на соглашение по написанию кода. Загрузка может быть в автоматическом режиме из xml файла, данный вид применяется, когда конфигурационные файлы уже загружались и пользователь не хочет их менять, в ручном режиме, когда необходимо изменить используемые конфигурационные файлы или нет доступа к интернету и в автоматическом режиме из интернета, в этом случае пользователю необходимо указать только *url*, откуда будет происходить загрузка. На рисунке 4 представлена диаграмма деятельности для данного прецедента. На ней мы видим возможные варианты загрузки конфигурационных файлов.

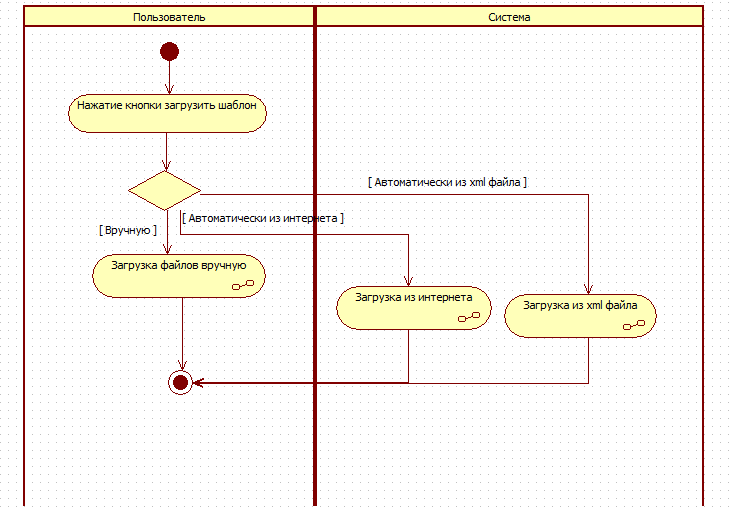


Рисунок 4 – Диаграмма деятельности прецедента «Загрузка конфигурационных файлов»

**Загрузка из xml файла**

На рисунке 5 представлена диаграмма деятельности для данного прецедента. Система пытается получить пути загружаемых файлов из xml файла с настройками. Если этих путей нет или они некорректные, то будет выдано сообщение об ошибке, и загружать файлы придется в ручном режиме.

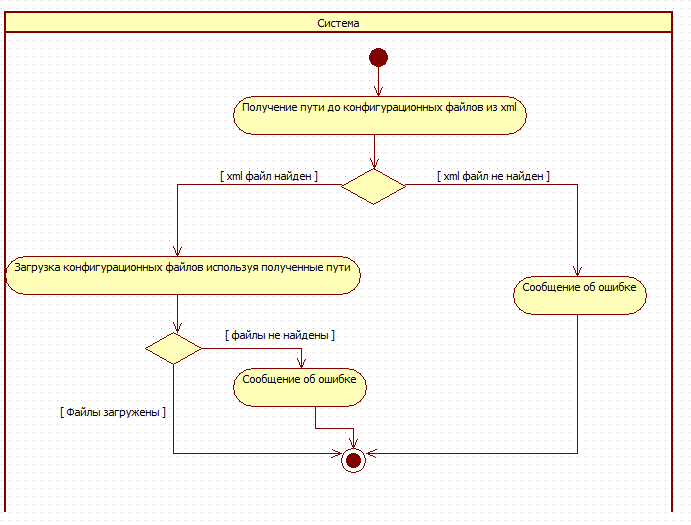


Рисунок 5 – Диаграмма деятельности прецедента «Загрузка из xml файла»

**Загрузка файлов вручную**

На рисунке 6 мы можем видеть, каким образом происходит процесс загрузки конфигурационных файлов вручную. Для начала пользователь должен нажать кнопку «Загрузка шаблона» и система предложит выбрать необходимый файл. Пользователь в этот момент может передумать и отменить загрузку. Далее необходимо загрузить конфигурацию для *checkstyle* по такому же алгоритму.

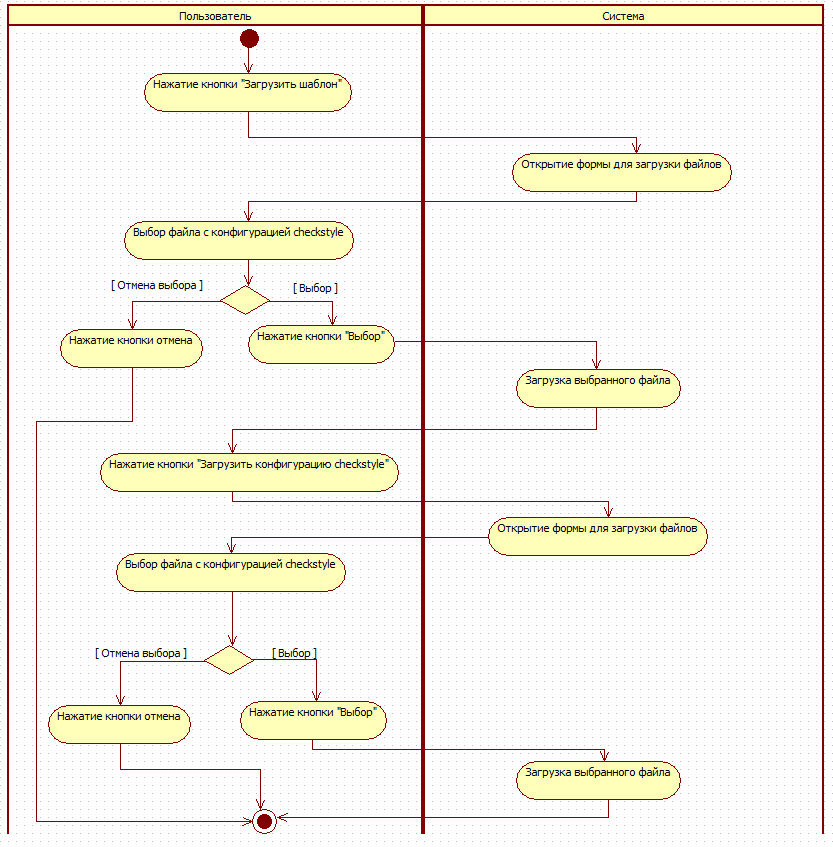


Рисунок 6 – Диаграмма деятельности прецедента «Загрузка файлов вручную»

**Загрузка файлов из интернета**

На рисунке 7 представлена диаграмма деятельности, которая отражает процесс загрузки конфигурационных файлов из интернета. Для этого пользователю необходимо указать *url* ссылку на интернет ресурс, откуда система может закачать файлы, в случае неудачи выведется сообщение об ошибке.

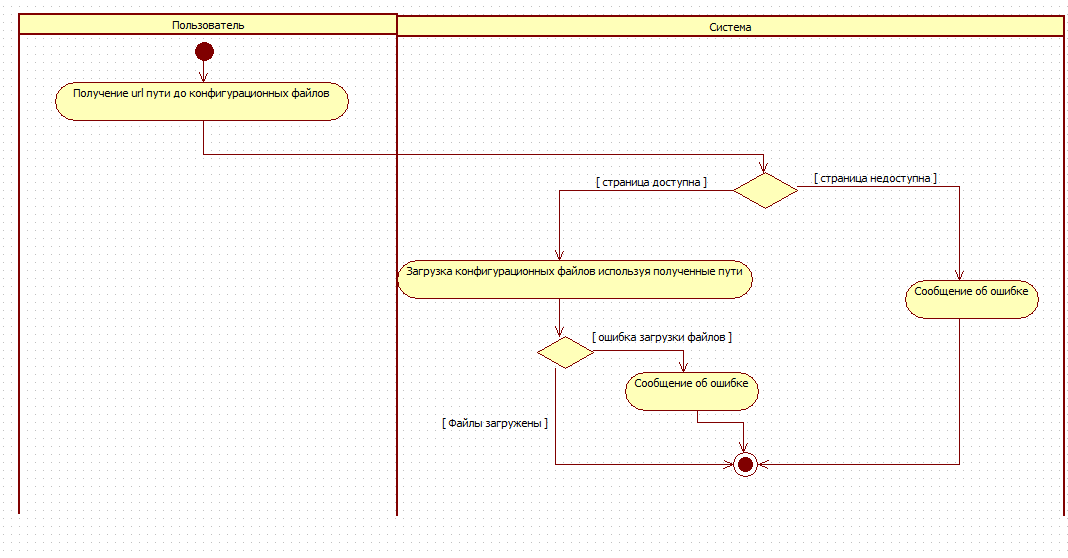


Рисунок 7 – Диаграмма деятельности прецедента «Загрузка файлов из интернета»

**Загрузка архива с проектом**

Загрузка архива происходит только в ручном режиме. Пользователь каждый раз должен выбирать архив с проектом. На рисунке 8 можно увидеть процесс загрузки архива в программу.

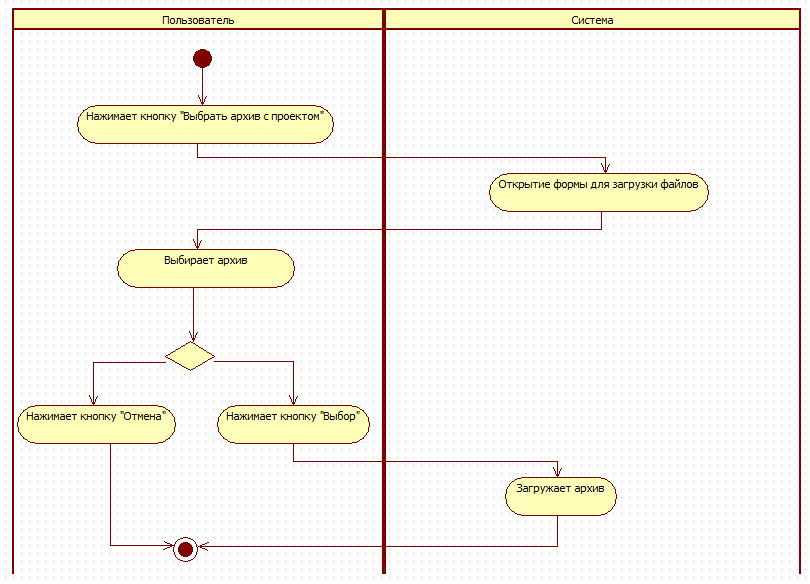
****

Рисунок 8 – Диаграмма деятельности прецедента «Загрузка архива с проектом»

**Запуск**

Перед тем как запускать программу необходимо, чтобы все конфигурационные файлы и файл с архивом были загружены. После этого программа дает возможность запустить свою работу. На рисунке 9 отражено, что происходит в процессе работы внутри системы. Сначала происходит построение иерархии папок, чтобы пользователь смог выбрать, какие файлы ему нужны в отчете. Далее происходит валидация исходных файлов и выдача результата.

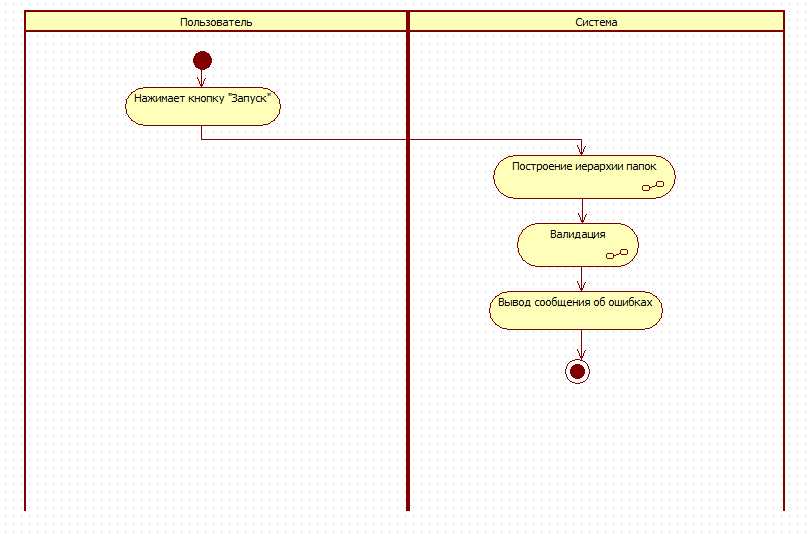
****

Рисунок 9 – Диаграмма деятельности прецедента «Запуск»

**Построение иерархии папок**

В процессе построения иерархии внутри системы происходит распаковка архива. После этого в памяти строится внутреннее представление иерархии файлов и папок. И далее это внутреннее представление графически отображается. Чтобы пользователь мог выбрать содержимое, каких файлов необходимо вставлять в отчет. Данный процесс представлен на рисунке 10.

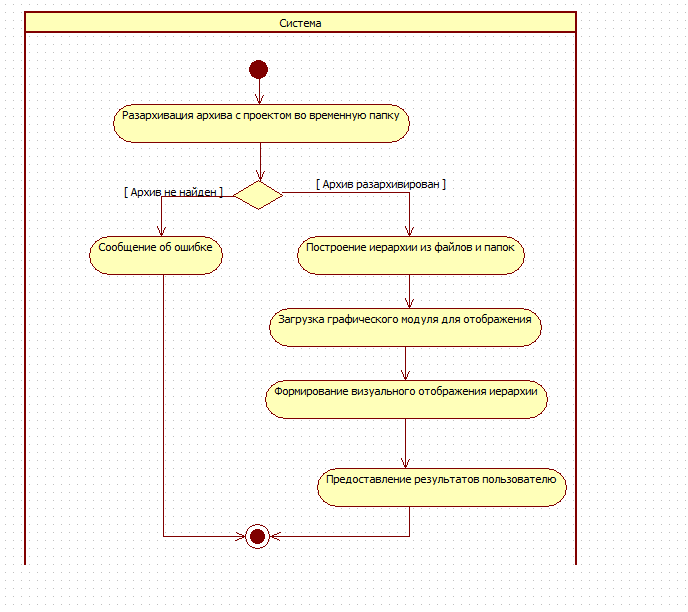
****

Рисунок 10 – Диаграмма деятельности прецедента «Построение иерархии папок»

**Валидация**

На рисунке 11 представлен процесс валидации исходных файлов. Для каждого исходного файла строится абстрактное синтаксическое дерево. Далее из конфигурационного файла достаются все проверки и далее эти проверки применяются к каждому абстрактному синтаксическому дереву. Сообщения об ошибках записываются в окно вывода.

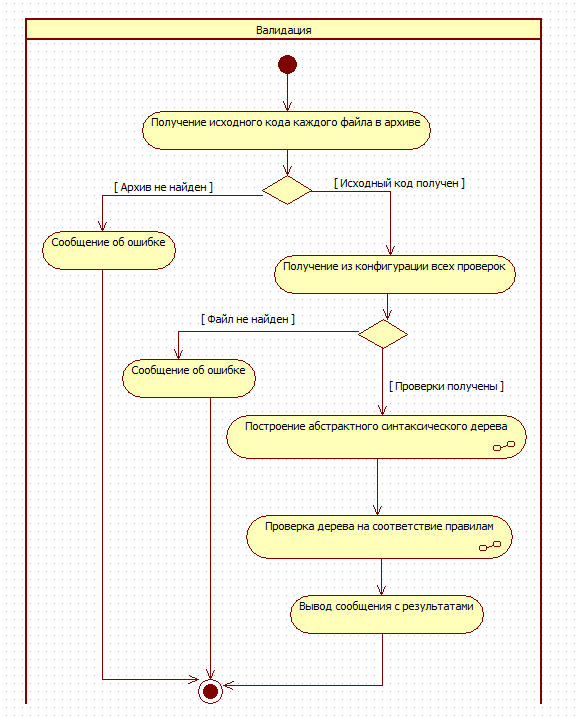
****

Рисунок 11 – Диаграмма деятельности прецедента «Валидация»

**Проверка дерева на соответствие правилам**

На рисунке 12 показано, как система производит проверку на дерева на соответствие правилам.

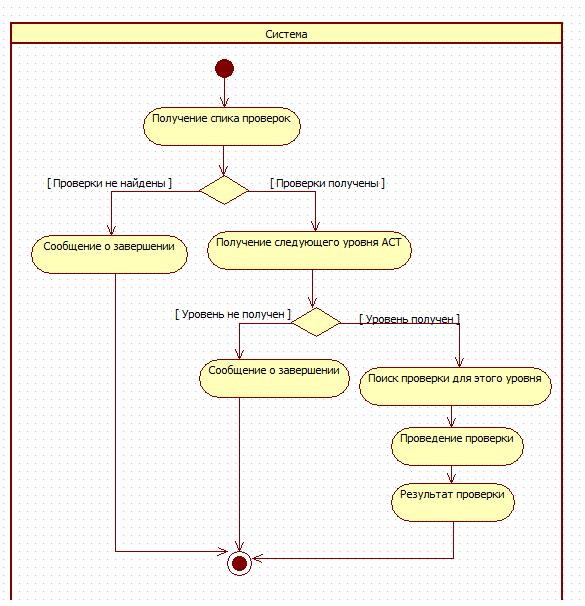
****

Рисунок 12 – Диаграмма деятельности прецедента «Проверка дерева на соответствие правилам»

**Формирование отчета**

На рисунке 13 показан процесс формирования отчета. Для того чтобы система могла сформировать отчет ей необходимо указать данные о студенте.

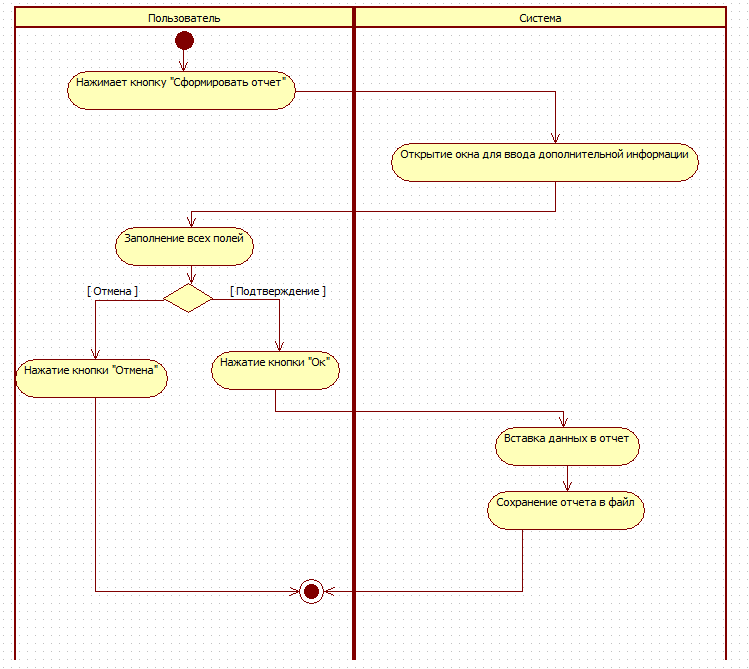
****

Рисунок 13 – Диаграмма деятельности прецедента «Формирование отчета»

**Выбор папок для включения в отчет**

Нет необходимости вставлять содержимое все файлов в отчет. Необходимо скопировать только наиболее важные участки исходного кода. Этот процесс идет после того, как архив с проектом был загружен и построена иерархия файлов и папок. На рисунке 14 отражен этот процесс.

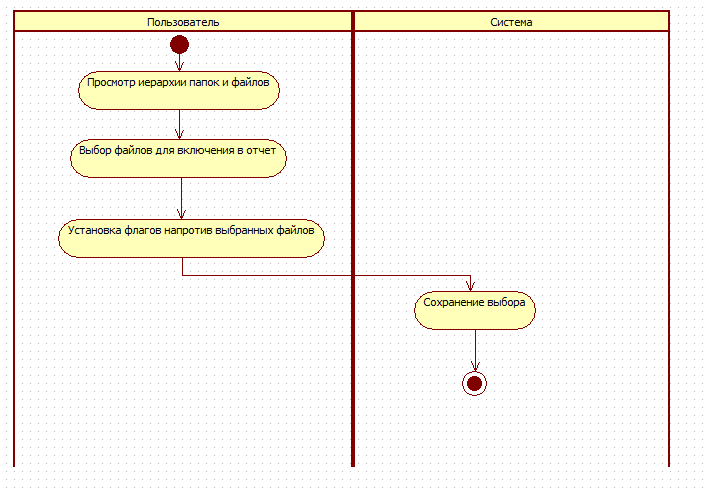
****

Рисунок 14 – Диаграмма деятельности прецедента «Выбор папок для включения в отчет»

**Формирование отчета**

Процесс формирования отчета является одним из главных и самых важных в работе программной системы. Он показан на рисунке 15.

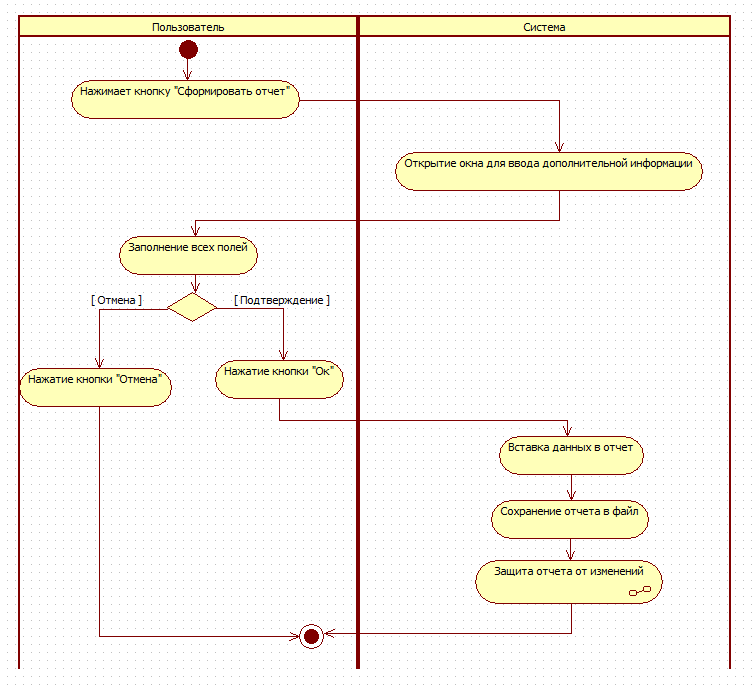
****

Рисунок 15 – Диаграмма деятельности прецедента «Формирование отчета»

**Защита отчета от изменений**

Данный прецедент нужен для того, чтобы преподаватель был уверен, что сформированный отчет не подвергался изменению. Так как отчет содержит в себе информацию о результатах проверки статическим анализатором кода. И студент может попытаться изменить ее. Диаграмма деятельности для данного прецедента показана на рисунке 16.

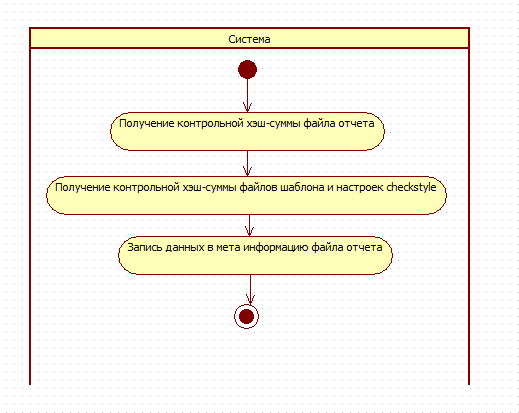
****

Рисунок 16 – Диаграмма деятельности прецедента «Защита отчета от изменений»

В результате анализа предметной области построена диаграмма прецедентов, на которой отражен весь основной функционал системы. Каждый прецедент имеет подробное описание и диаграмму деятельности.

# Разработка алгоритмов работы системы

Перед тем как переходить к непосредственно разработке [20] программной системы необходимо продумать и обосновать алгоритмы работы отдельных наиболее важных участков системы. Для этого сформируем базовую структуру программной системы автоматизации отчетов по лабораторным работам. Она представлена на рисунке 17.

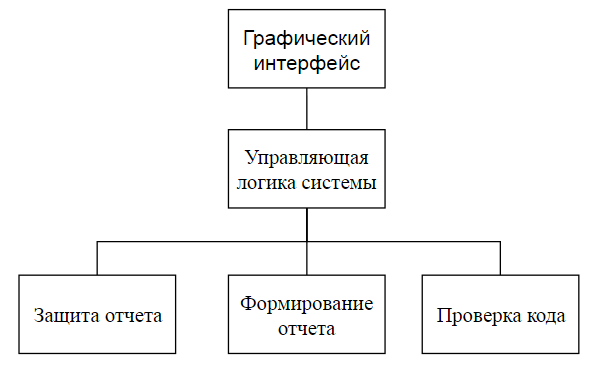


Рисунок 17 – Структура программной системы

Использование модульной структуры позволит использовать уже написанный код повторно. Например, модуль формирования отчета можно будет использовать как плагин для других программных систем. Модуль «Проверка кода» отвечает за проверку качества исходного кода. В него будут включены различные статические анализаторы кода, которые будут выдавать результат проверки. Модуль «Формирование отчета» отвечает непосредственно за формирование отчета. Он использует в качестве входных данных шаблон с метками. Эти метки необходимо заменить на соответствующую информацию. Например, могут быть метки для вставки исходного кода, результата проверки исходного кода статическими анализаторами, информации о студенте. Эти данные будут проходить валидацию, чтобы в поле, которое предусматривает вставку чисел, нельзя было поместить строку. Шаблон представляет собой текстовый документ в формате *rtf*. Отчет также сохраняется как текстовый документ. Модуль «Защита отчета» будет производить генерацию контрольной суммы, для обеспечения защиты отчета от изменений. Для того чтобы, пользователь мог взаимодействовать и системой необходим графический интерфейс. Он должен отражать все функции и возможности системы. И выдавать всю необходимую информацию.

На рисунке 18 представлен общий алгоритм работы программной системы.

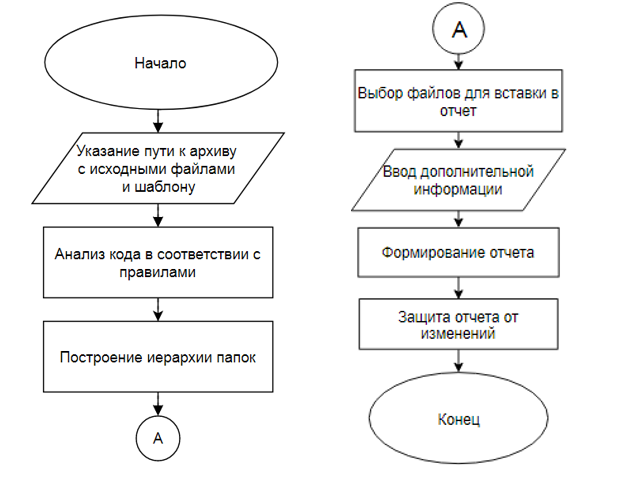


Рисунок 18 – Общий алгоритм работы программной системы

Пользователю необходимо указать пути к архиву с исходными файлами проекта и шаблону, по которому будет формироваться отчет. Далее будет произведен анализ исходного кода с помощью статических анализаторов кода. Далее система строит иерархию папок и файлов из полученного архива. Это необходимо для того, чтобы пользователь мог выбрать исходный код каких именно файлов необходимо вставить в отчет. Ведь в листинг обычно вставляется, только наиболее важные и значимые участки кода. После этого система должна проанализировать метки, которые есть в шаблоне, и запросить в соответствии с ними дополнительную информацию у пользователя. После этого вся полученная информация будет вставлена в ответ и произойдет его формирование. Далее полученный отчет будет необходимо защитить с помощью получения контрольной суммы.

На рисунке 19 представлен алгоритм анализа кода в соответствии с заданными правилами. Система берет все файлы, которые содержат исходный код и для каждого строит абстрактное синтаксическое дерево (АСД) [18]. После этого начинается анализ полученного АСД, для этого берется каждый узел дерева и происходит проверка, есть ли правила в конфигурационном файле, которые подходят для данного узла. Если такие правила находятся, то они применяются. Если нет, то осуществляется переход до следующего узла АСД. Этот процесс повторяется до тех пор, пока не будет проверенно все дерево. Только после этого будет сделан переход к следующему файлу. Вся информация о результатах хранится в памяти. И после завершения проверки выводится на экран, для того, чтобы пользователь мог с ней ознакомиться. Если архив с проектом не содержит файлов с исходным кодом, то система не будет производить никаких проверок, а сразу перейдет к следующему этапу. Для определения типа файла приложение анализирует его расширение.

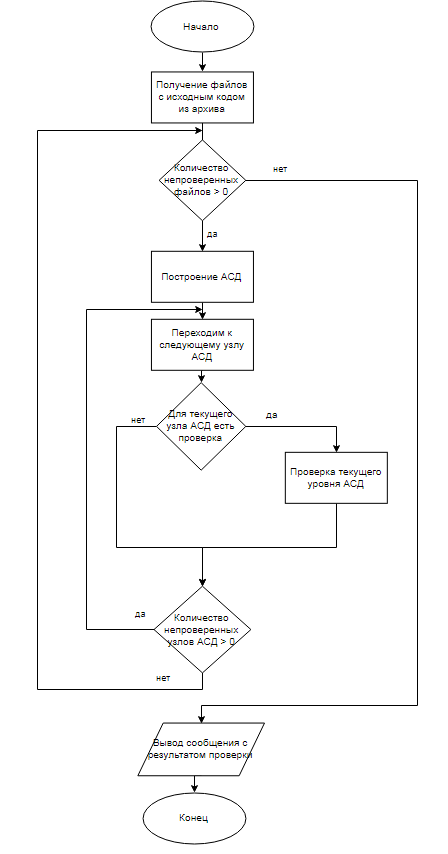


Рисунок 19 – Алгоритм работы анализа кода

На рисунке 20 продемонстрирован алгоритм запроса дополнительной информации у пользователя и формирования отчета. Вначале шаблон отчета анализируется, и система получает все метки, которые там содержаться. Затем происходит уточнение информации у пользователя в соответствии с полученными метками. После этого полученная информация проверяется, чтобы не получилось так, чтобы в поле предусматривающее число была записана строка. Далее отчет дополняется результатами проверки кода и исходным кодом выбранных файлов. После этого отчет сохраняется в формате *rtf*.

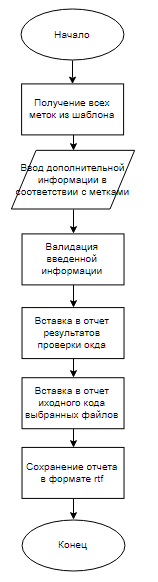


Рисунок 20 – Алгоритм формирования отчета

Алгоритм защиты отчета от изменений продемонстрирован на рисунке 21. Вычисление контрольной суммы для файлов шаблона и конфигурационного файла для анализа кода необходимо, чтобы преподаватель был уверен, что использовались верные шаблон и настройки для статического анализатора кода. А вычисление контрольной суммы отчета позволяет защитить отчет от изменений. При изменении файла с отчетом мета информация изменится, и контрольная сумма будет утрачена. И если контрольную сумму оставить в мета информации, то при проверке преподавателем контрольные суммы не сойдутся.

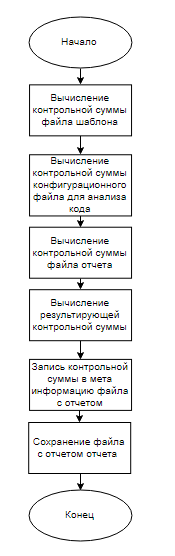


Рисунок 21 – Алгоритм защиты отчета от изменений

В результате разработки алгоритмов программная система разбита на модули, созданы алгоритмы для всех основных функций и выбраны шаблоны проектирования, которые необходимо использовать в процессе разработки.

# Разработка программной системы

В ходе разработки программной системы необходимо разработать и реализовать пользовательский интерфейс [9] и так же реализовать программно все функции системы.

## Разработка пользовательского интерфейса

Интерфейс пользователя должен обеспечивать доступ ко всем функциям системы, полностью отображать всю выводимую информацию и быть удобным для использования.

На рисунке 22 представлено шаблон главного окна программы, которое появляется после ее запуска.

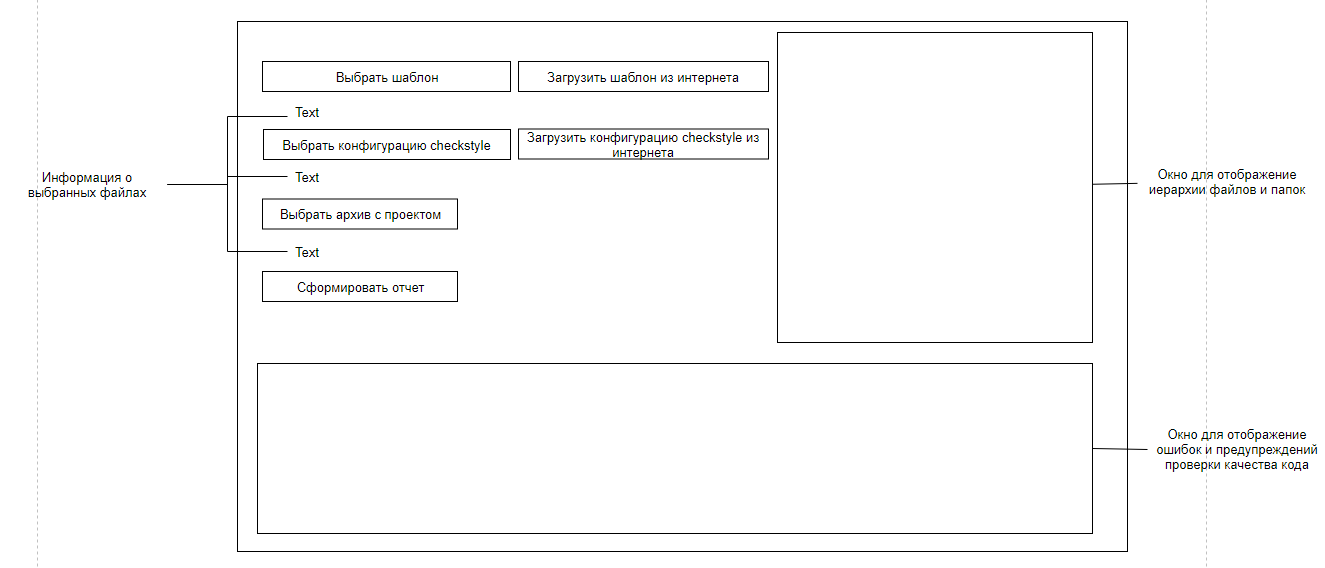


Рисунок 22 – Макет главного окна программы

При нажатии на одну из кнопок загрузки должно появляться окно с выбором файлов. При нажатии на кнопки загрузки из интернета должно появляться окно, в котором пользователь будет вводить *url* адрес. Макет данного окна представлен на рисунке 23. Если же файл уже был загружен, то информация о его имени должна быть под кнопкой загрузки, чтобы пользователь знал, какой файл был загружен и где на компьютере он расположен.

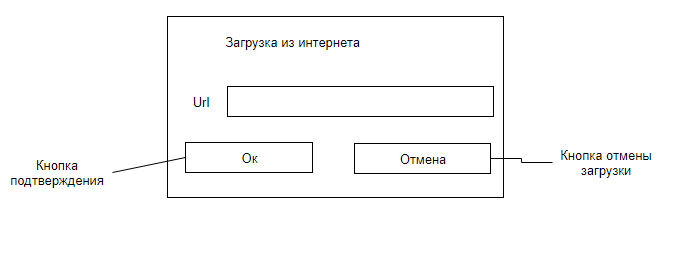


Рисунок 23 – Макет окна загрузки файлов из интернета

Окно с иерархией файлов и папок должно предоставлять возможность по чекбоксу выбирать не только отдельный файл, но и всю папку с содержимым целиком. Это позволит пользователю делать более быстрый выбор, если надо выбрать сразу много файлов. На рисунке 24 представлен шаблон этого окна.

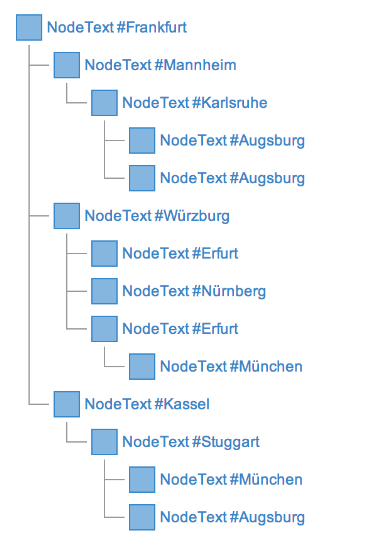


Рисунок 24 – Макет окна с иерархией файлов и папок

При генерации отчета программа должна выдавать окна с просьбой ввода дополнительной информации. Информация о том, какие поля необходимо выводить, берется из файла шаблона и динамически генерируется во время выполнения программы. Шаблон с окном для ввода представлен на рисунке 25.

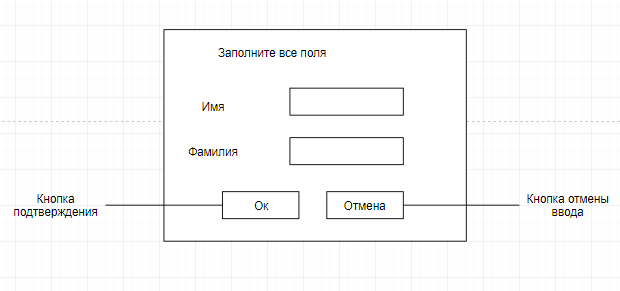


Рисунок 25 – Макет окна с информацией о студенте

На рисунке 26 представлен макет окна для проверки подписи отчетов. Для проверки необходимо указать папки с шаблоном, конфигурацией *checkstyle*, архивом с проектом и отчетом.

В результате разработки пользовательского интерфейса спроектированы макеты всех окон программной системы.

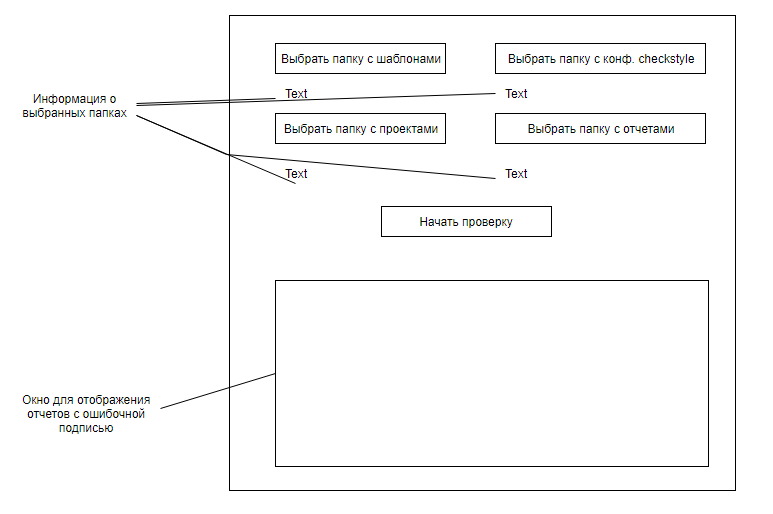


Рисунок 26 – Макет окна для проверки подписи отчетов

## Реализация пользовательского интерфейса

После того, как пользовательский интерфейс был спроектирован, его необходимо реализовать. Для этого мы используем *JavaFX*, это позволит нам быстро сконфигурировать нужный нам интерфейс.

На рисунке 27 представлено главное окно программной системы.

На рисунке 28 представлено главное окно программы после загрузки файла с шаблоном и конфигурацией *checkstyle*. Так же мы видим пути, по которым расположены данные файлы.

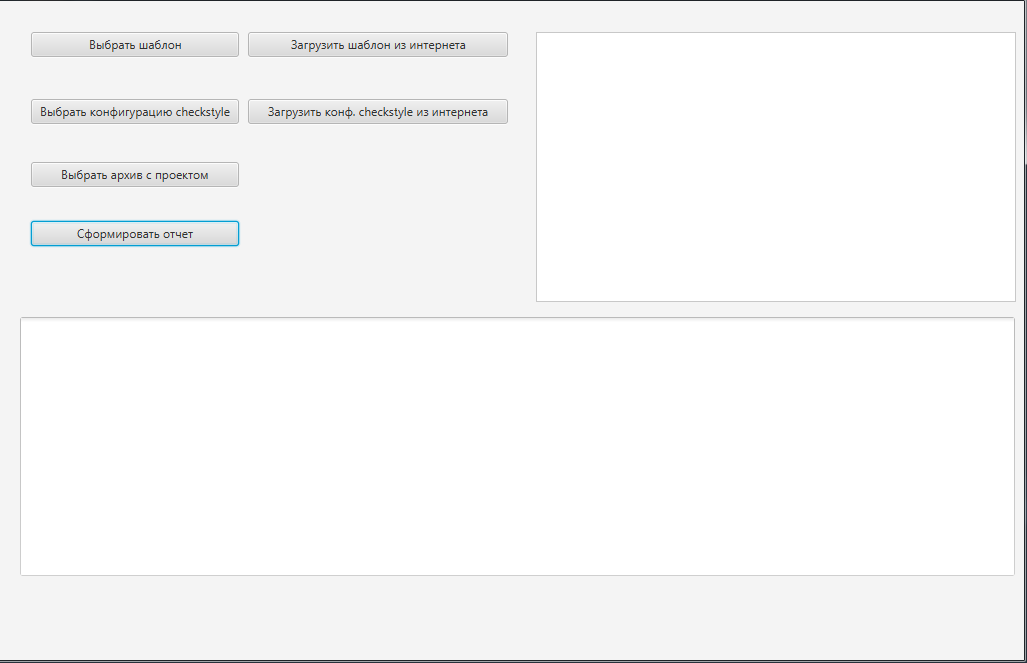


Рисунок 27 – Главное окно программы

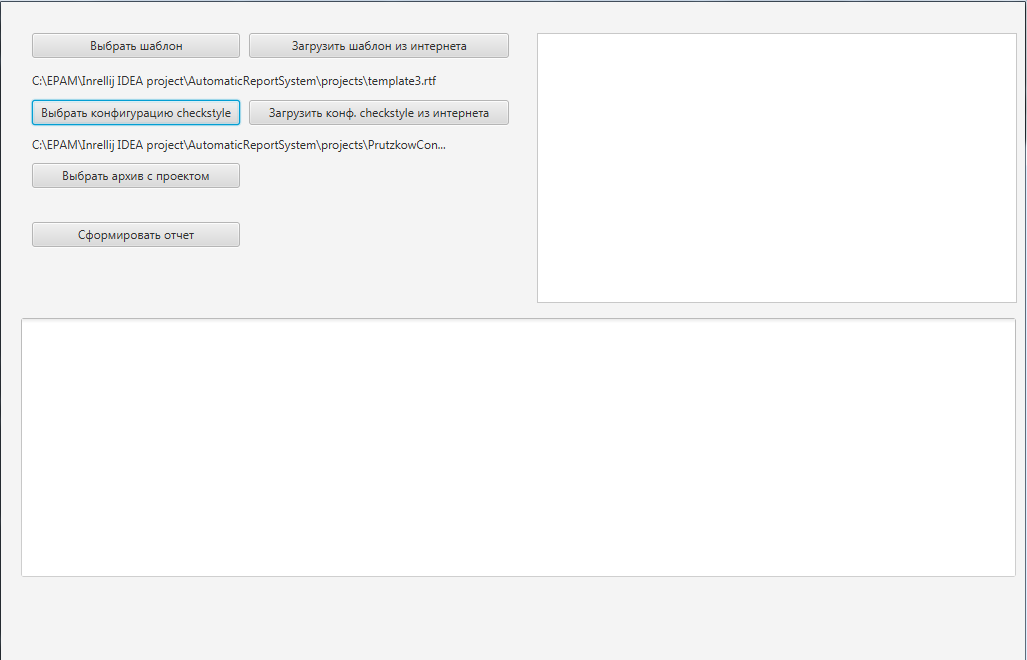


Рисунок 28 – Главное окно программы после загрузки файлов шаблона и конфигурации *checkstyle*

На рисунке 29 представлено главное окно, после того как был загружен архив. Видно, что в правой части окна сформировалась иерархия файлов и папок. А в нижней части выдан результат проверки файлов с исходным кодом с помощью статического анализатора кода.

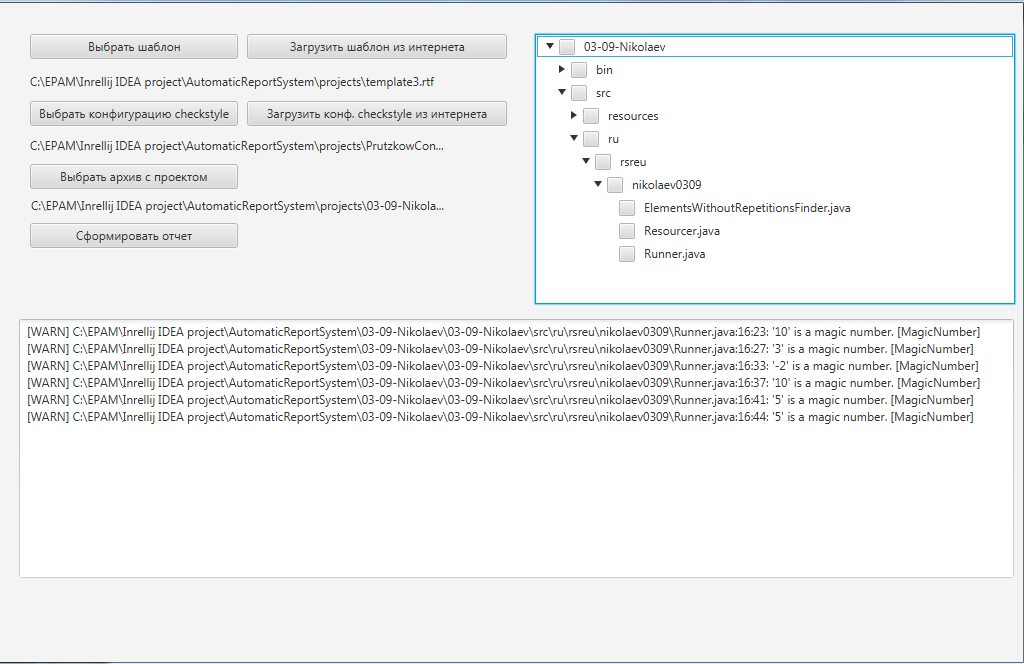


Рисунок 29 – Главное окно после загрузки архива с проектом

На рисунке 30 представлен процесс выбора файлов и папок для вставки в отчет. Видно, что можно выбирать как отдельные файлы, так и папки целиком.

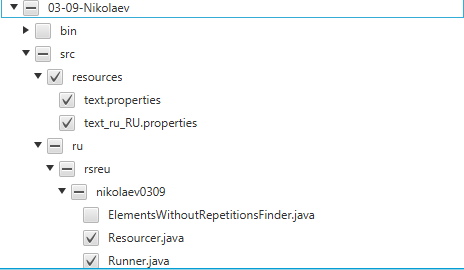


Рисунок 30 – Выбор файлов

На рисунке 31 представлен результат проверки файлов с исходным кодом статическим анализаторов кода *checkstyle*. В информации отражается уровень ошибки, файл и номер строки, где ошибка была найдена. А так же сообщение к ошибке.

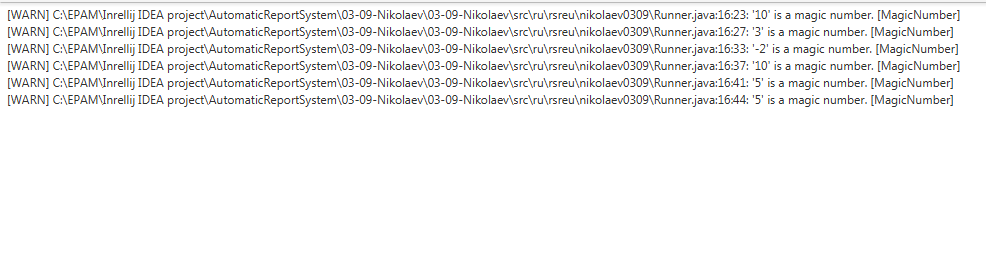


Рисунок 31 – Найденные ошибки *checkstyle*

На рисунке 32 показано окно с вводом дополнительной информации о пользователе.

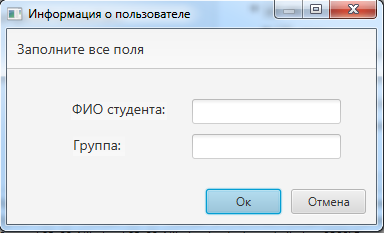


Рисунок 32 – Ввод дополнительной информации

На рисунке 33 представлено окно для проверки подписи отчетов. С его помощью преподаватель может убедиться в том, что отчет составлен с использованием правильного шаблона и конфигурации checkstyle, а также не изменялся.

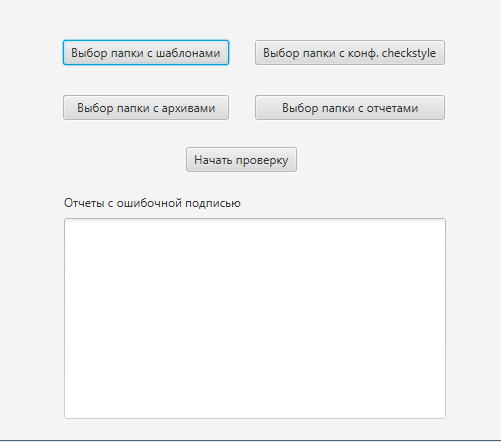


Рисунок 33 – Окно проверки подписи отчетов

В результате реализации пользовательского интерфейса созданы все окна программной системы. Для реализации использовалась библиотека *JavaFX.*

## Структура программной системы

В ходе реализации программной системы использовался шаблон *MVC*  (Модель-Представление-Контроллер) [19, 29]. Данный паттерн используется для разделения данных приложения, пользовательского интерфейса и управляющей логики. Это позволяет писать более гибкие приложения. Использования этого шаблона в программной системе позволит нам при необходимости заменить интерфейс пользователя или перенести все в веб-приложение [13], без изменения модели.

На рисунке 34 представлена структура пакетов системы. Все классы распределены по пакетам в зависимости от их назначения.

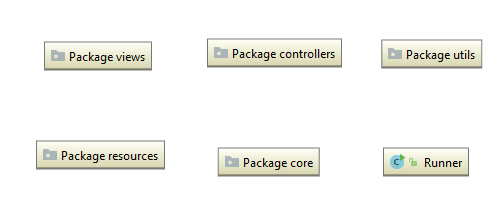


Рисунок 34 – Структура пакетов системы

Точка входа приложения – это класс *Runner*. С него начинается работа, и он запускает необходимые представления. В пакете *views* находятся представления пользовательского интерфейса. Так как используется *JavaFX*, то все представления определяются как файлы с расширением *fxml*, например *MainView.fxml*. В пакете *controllers* находятся вся управляющая логика приложения, которая позволяет связать нам представление и модель, которая находится в пакете *core*. В пакете resources находятся файлы для работы приложения, например сообщения, которые будут выводиться для различных меток в шаблоне. В пакете *utils* находятся различные утилиты, для более удобной работы с файлами и ресурсами.

На рисунке 35 представлена диаграмма классов [4] из пакета *controllers*. Данные классы позволяют нам обрабатывать события, которые приходят от пользовательского интерфейса, отправляя их на модель. После того, как модель изменилась, меняется и представление. Для каждого элемента пользовательского интерфейса имеется свой контроллер. Это необходимо, чтобы не делать один класс, в котором будут находиться все обработчики, так как такой класс очень трудно изменять и понимать, что в нем происходит.

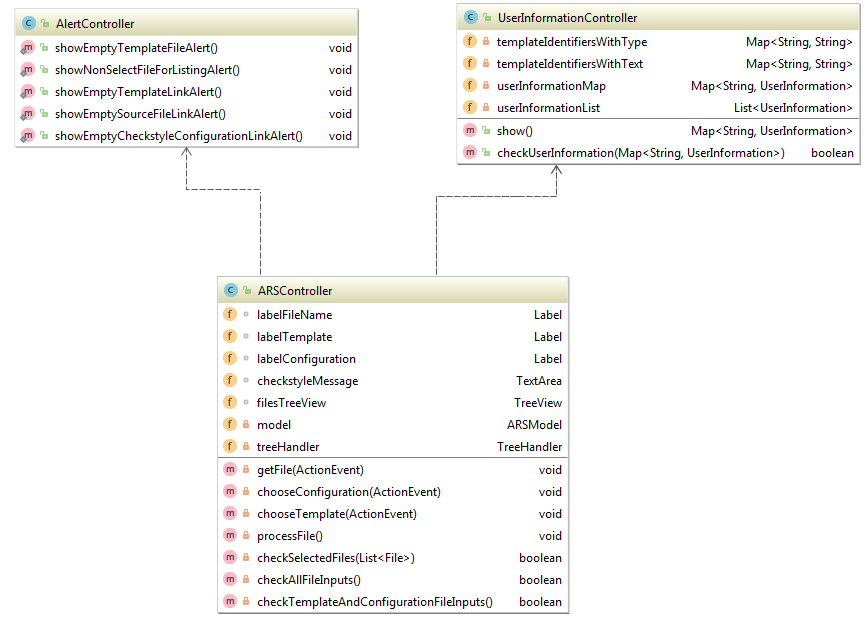


Рисунок 35 – Диаграмма классов пакета controllers

На рисунке 36 представлена диаграмма классов, которые располагаются в пакете core. В этом пакете располагается модель. Аннотация *RunCheck* необходимо для того, чтобы пометить какие методы должны вызваться в классе *CodeChecker* [32]. Для того чтобы добавлять проверки с использованием других статических анализаторов кода, не изменяя при этом код, а только лишь расширяя класс *CodeChecker* новыми методами, помеченными аннотацией *RunCheck*. Модуль защиты отчета от изменения реализован в классе Security, он позволяет получать контрольную сумму. Этот же класс используется при проверке подписи отчетов. Данная проверка реализована в отдельном, небольшом приложении, которое используется преподавателем.

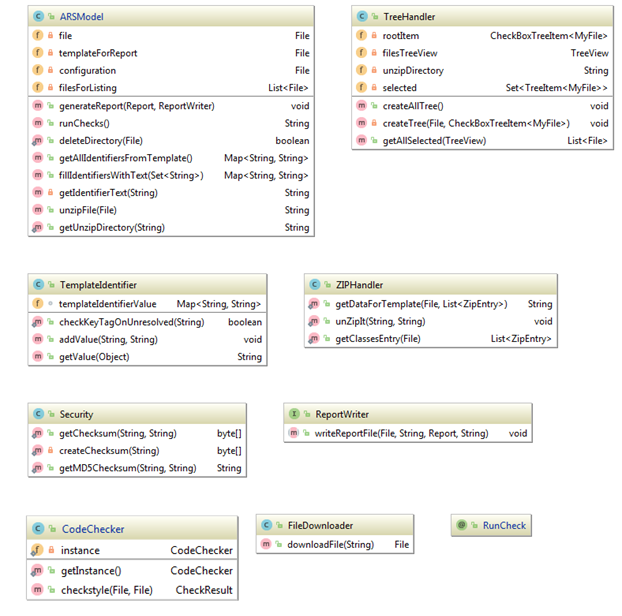


Рисунок 36 – Диаграмма классов пакета core

В ходе разработке использовались такие порождающие шаблоны, как фабрика и одиночка [25]. Шаблон фабрика реализован в классе *FileChooserConfigurationFactory*. Он необходим для того, чтобы вынести процесс создания настроек для выбора файлов. Этот процесс повторяется несколько раз и подразумевает использование одного и того же кода. Шаблон одиночка реализован в классе *CodeChecker*. И сделано это для того, чтобы не иметь нескольких экземпляров этого класса. В работе программной системы объект этого класса инициализируется только один раз. В программной системе используется структурный шаблон адаптер. Он реализован в классе *MyFile*. Он используется как класс обертка для стандартной реализации *File*. Это позволяет использовать этот класс при построении дерева иерархии файлов и папок. Для использования упрощенного интерфейса в процессе создания и проверке отчетов используется структурный шаблон фасад.

Диаграмма компонентов и диаграмма развёртывания образуют группу диаграмм реализации. На рисунках 37-41 представлена модель исходного кода программной системы с помощью диаграммы компонентов. Из-за спецификации *Java* каждый класс собирается в файл с таким же именем, но расширением *.java*. Далее каждый этот файл интерпретируется в *.class*. И уже на основе *\*.class* файлов собирается проект. Из-за этой тонкости диаграммы получаются обширными.

На рисунке 42 представлена модель исполняемого программного кода, которая представляет с помощью стереотипа расширения «*build*» трансформацию исходного кода в исполняемый код, т. е. показывает взаимосвязь между одними и теми же артефактами, представленными в разных физических форматах.

На рисунке 43 представлена взаимосвязь артефактов через интерфейс.

Для того, чтобы отобразить размещений артефактов на физическом устройстве используется диаграммы развертывания. Данная диаграмма представлена на рисунке 44.

В результате разработки структуры программной системы построены диаграммы классов для основных модулей системы. Описана реализация каждого модуля, построены диаграммы компонентов и развертывания.

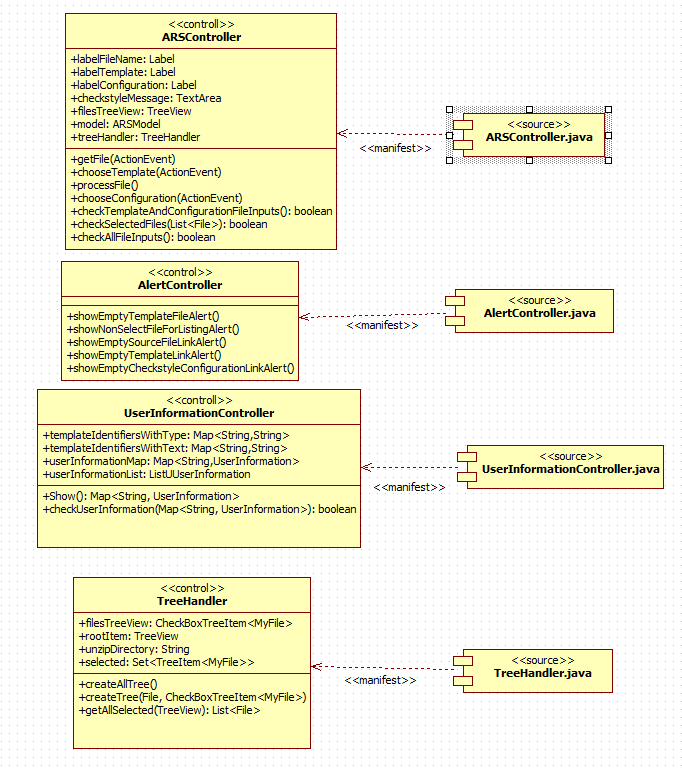


Рисунок 37 – Диаграмма компонентов: модель исходного кода для системы часть 1

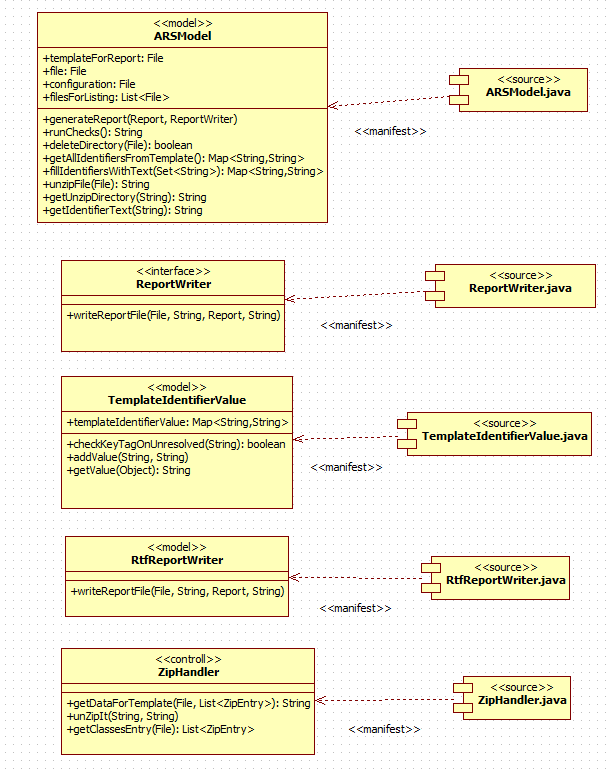


Рисунок 38 – Диаграмма компонентов: модель исходного кода для системы часть 2

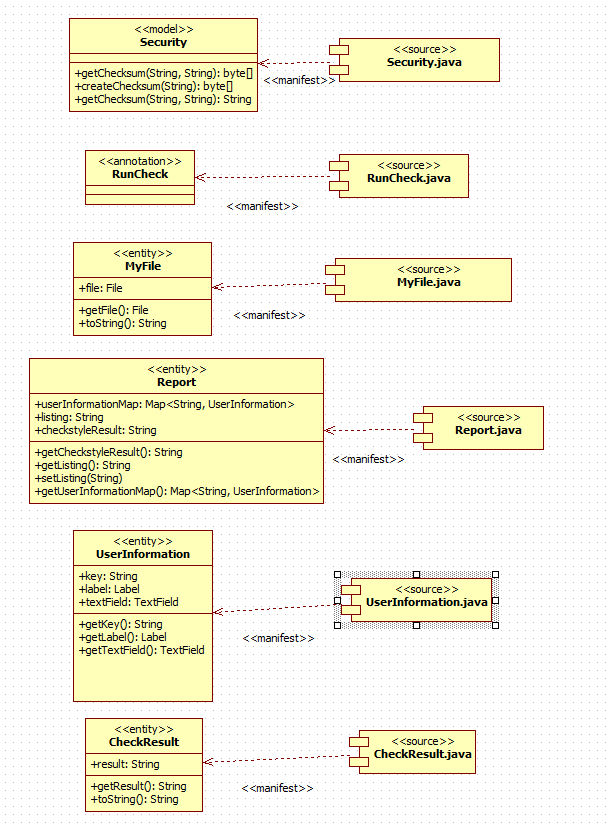


Рисунок 39 – Диаграмма компонентов: модель исходного кода для системы часть 3

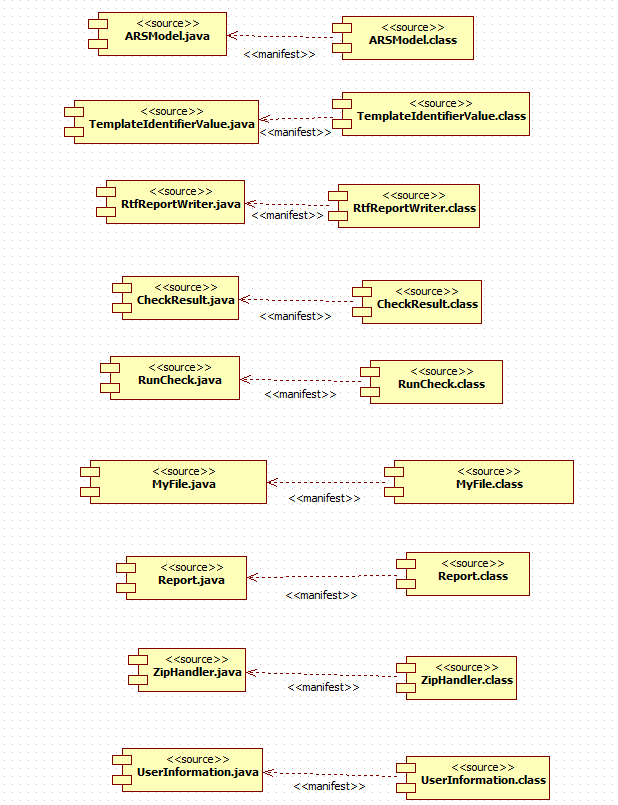


Рисунок 40 – Диаграмма компонентов: модель исходного кода для системы часть 4

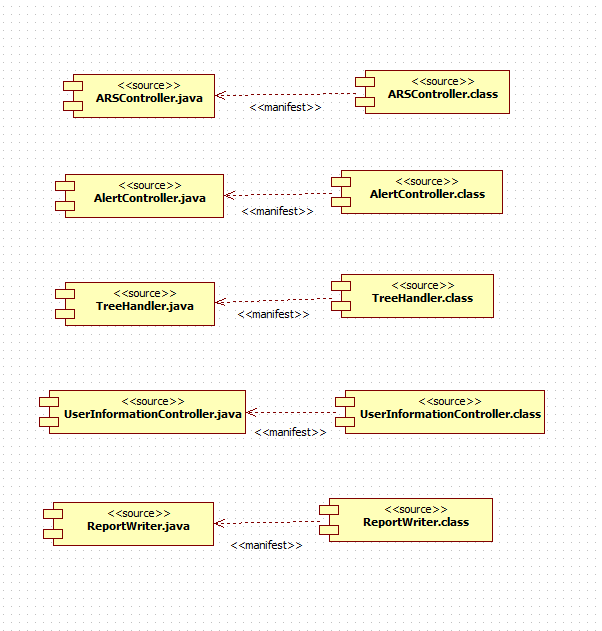


Рисунок 41 – Диаграмма компонентов: модель исходного кода для системы часть 5

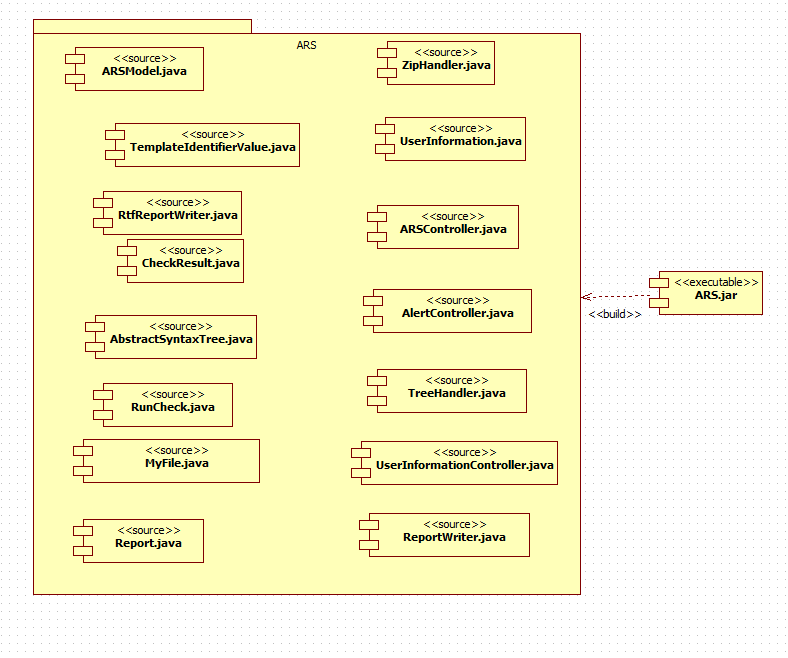


Рисунок 42 – Диаграмма компонентов: модель исполняемого программного кода

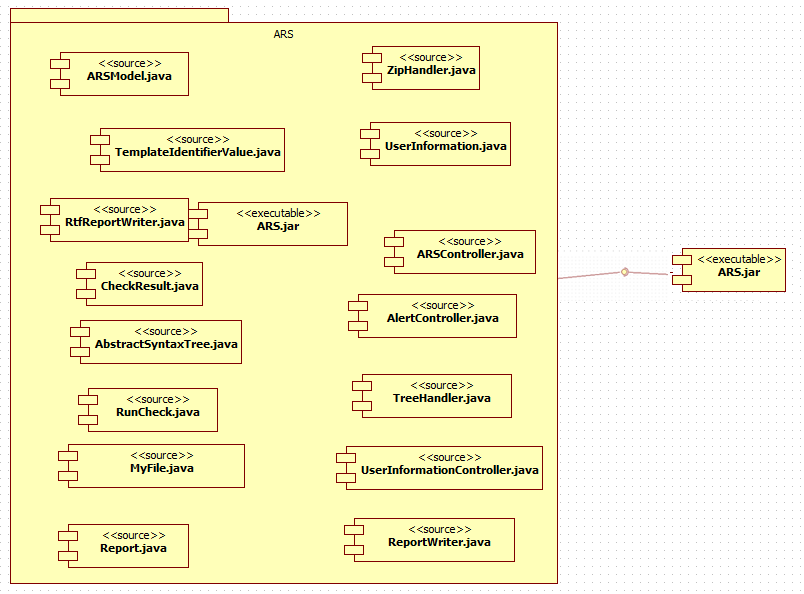


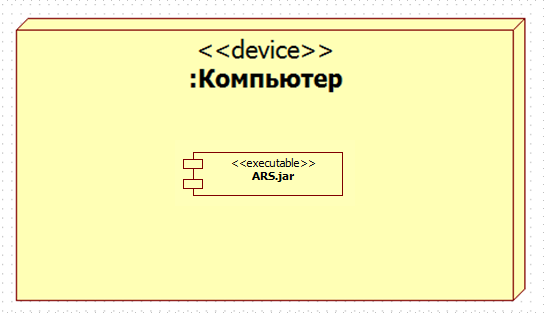
Рисунок 43 – Диаграмма компонентов: модель артефактов системы

Рисунок 44 – Диаграмма развертывания

# Разработка программной документации

При разработке программной системы необходимо грамотно составить документацию по ней. Она поможет программисту вносить изменения в функционал системы, тратя на это меньше времени. Документация поможет пользователю быстрее разобраться в работе программной системы.

## Руководство программиста

Программная система может быть расширена [1] и к ней может добавляться новый функционал. Для того, чтобы изменить формат в котором будет сохранен отчет необходимо расширить интерфейс *ReportWriter*

**public interface** ReportWriter {  
 **void** writeReportFile(File templateForReport, String outputFileName, Report report);  
}

Реализовать необходимо метод *writeReportFile (File templateForReport, String outputFileName, Report report)*, который в качестве первого аргумента принимает файл с шаблоном отчета и данные для отчета в качестве второго. После этого необходимо экземпляр получившегося класса передать в качестве второго параметра в метод *generateReport (Report report, ReportWriter reportWriter) из класса ARSModel*. Таким образом, можно сохранять отчет в любом необходимом формате или загружать в интернет.

Программную систему можно расширить, добавив дополнительные проверки для кода. Для этого в классе *CodeChecker* необходимо написать методы для проверки кода. Новые методы для проверки должны принимать файл архива в качестве первого аргумента и файл с конфигурацией проверки в качестве второго аргумента. При этом возвращаемое значение должно быть типа *CheckResult*, в объект этого класса можно записать всю информацию о проведенной проверке. Методы, которые будут проверять код необходимо пометить аннотацией *RunCheck*, это позволит внедрить данные методы в систему и автоматически вызывать их в процессе работы для проверки содержимого архива с проектом.

Файлы шаблона и конфигурации статического анализатора кода можно загружать из различных источников, для того чтобы реализовать загрузку из нужного вам источника необходимо реализовать интерфейс *FileDownloader*

**public interface** FileDownloader {  
 File downloadFile(String name);  
}

Реализовать необходимо метод *downloadFile (String name),* который в качестве аргумента принимает имя файла, который необходимо скачать или любые другие идентификаторы данного файла.

При желании изменить алгоритм формирования контрольной суммы, необходимо изменить метод *createChecksum* в классе *Security*. Для изменения алгоритма шифрования, необходимо передать данному методу имя нового алгоритма.

Для того, чтобы изменить представление необходимо зайти в пакет *views* и открыть форму и необходимым представлением с помощью *Scene Builder*, если используется *IntelliJ IDEA*, или можно отредактировать этот файл вручную с помощью любого текстового редактора, так как он представлен, как *xml* документ. При добавлении новых элементов обработчики для них необходимо прописать в классе *ARSController*.

В результате составления руководства программиста описаны основные возможности по модификации системы, классы и интерфейсы, которые используются при реализации нового функционала.

## Руководство пользователя

Перед использованием необходимо сформировать шаблон для отчета. Шаблон для отчета представляет текстовый файл в формате *rtf*. Для вставки информации в шаблон в шаблоне используются метки. Формат метки следующий: *%%name#type%%.* Где *name* – это имя метки, по этому имени будет происходить вставка и будет запрошена дополнительная информация, а *type* – это тип метки, в соответствии с этим типом информация для этой метки будет проходить проверку на соответствие указанному типу. Разрешены следующие типы: *integer, float, string*. Метки *%%Code#string%%* и *%%checkstyle#string%%* предназначены для вставки листинга и результатов проверки статическими анализаторами кода соответственно. На рисунке 45 представлен пример отчета.

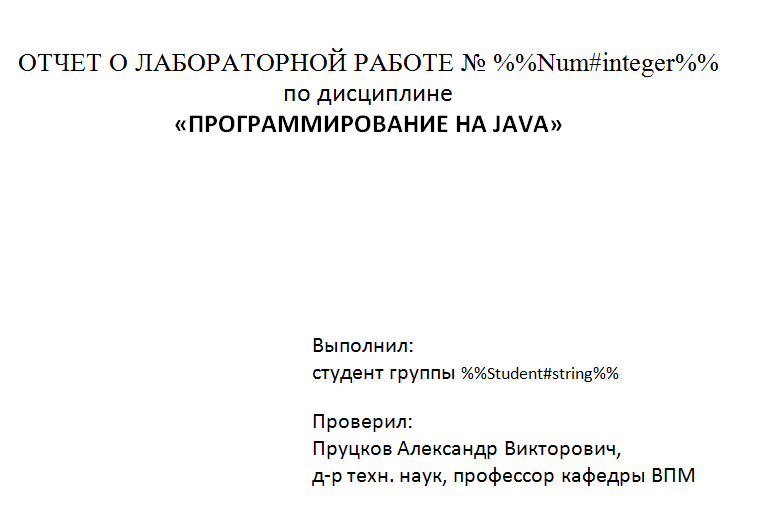


Рисунок 45 – Шаблон отчета

Для использования данной программной системы необходимо выполнить следующие действия:

1. Запустите приложение, это происходит путем двойного нажатия на файл приложения. При успешном запуске на экране появится главное окно приложения. Оно представлено на рисунке 27.
2. Укажите файл с шаблоном для отчета. Если приложение ранее уже запускалось, и шаблон для отчета уже выбирался, то он будет загружен автоматически. Если вы хотите изменить используемый файл с шаблоном отчета или он еще ни разу не выбирался, то для этого есть два варианта.

*Вариант 1.* Выбрать файл шаблона на компьютере вручную.

Нажмите на кнопку «Выбрать шаблон». Ее расположение представлено на рисунке 46.

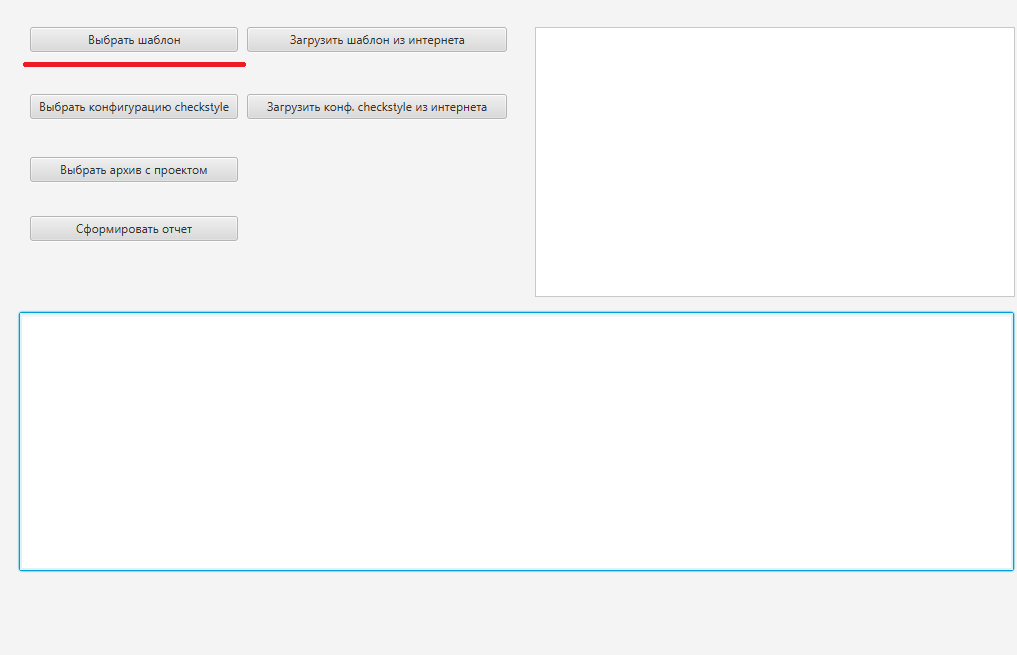


Рисунок 46 – Выбор файла с шаблоном вручную

На рисунке 47 представлено открывшееся диалоговое окно для выбора файла. Необходимо выбрать нужный шаблон и нажать кнопку «Открыть». Если было принято решение не загружать новый шаблон, необходимо нажать кнопку «Отмена».

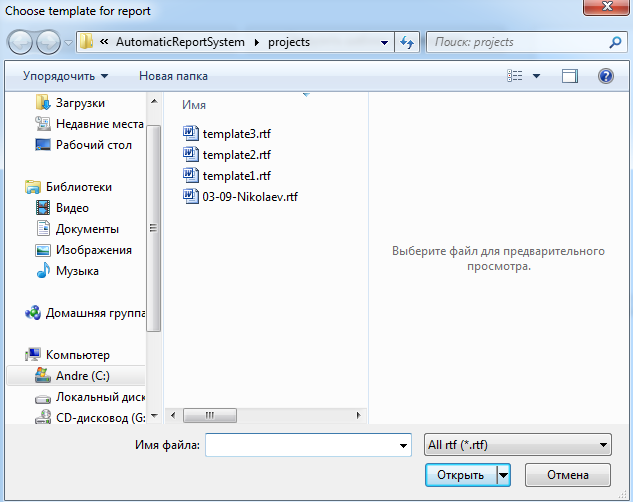


Рисунок 47 – Выбор шаблона отчета

*Вариант 2.* Загрузка файла шаблона из интернета.

Нажмите на кнопку «Загрузить шаблон из интернета». Ее расположение представлено на рисунке 48.

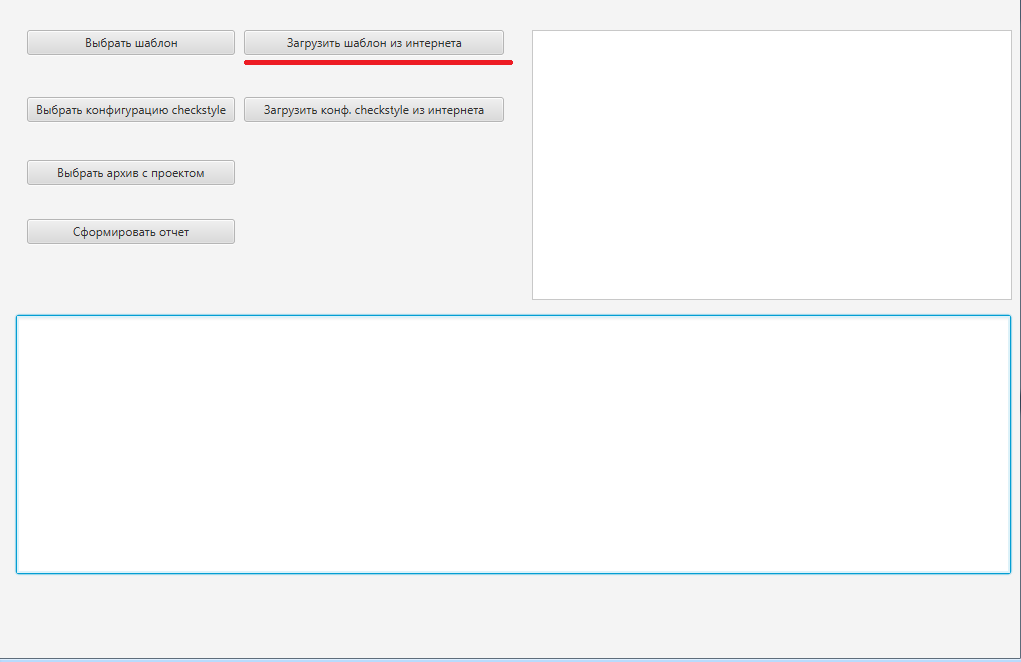


Рисунок 48 – Загрузка шаблона отчета из интернета

На рисунке 49 представлено открывшееся диалоговое окно для загрузки файла из интернета. Необходимо ввести в строку ввода *url* адрес и нажать кнопку «Загрузить шаблон». В случае если *url* будет неверным или окно ввода останется пустым. Данное окно откроется еще раз. Если было принято решение не загружать шаблон из интернета, необходимо нажать кнопку «Отмена».

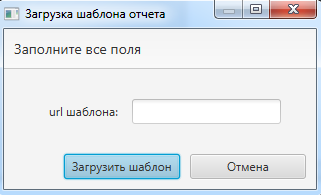


Рисунок 49 – Диалоговое окно ввода *url* для загрузки шаблона из интернета

На рисунке 50 представлен удачно загруженный шаблон. Под кнопками «Выбрать шаблон» и «Загрузить шаблон из интернета» появился путь до файла с шаблоном.

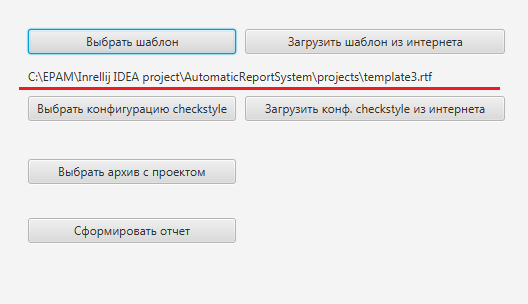


Рисунок 50 – Успешная загрузка шаблона

1. Укажите файл с конфигурацией для статического анализатора кода *checkstyle*. Если приложение ранее уже запускалось, и файл с конфигурацией уже выбирался, то он будет загружен автоматически. Если вы хотите изменить используемый файл с конфигурацией или он еще ни разу не выбирался, то для этого есть два варианта.

*Вариант 1.* Выбрать файл конфигурации *checkstyle* на компьютере вручную.

Нажмите на кнопку «Выбрать конфигурацию *checkstyle*». Ее расположение представлено на рисунке 51.

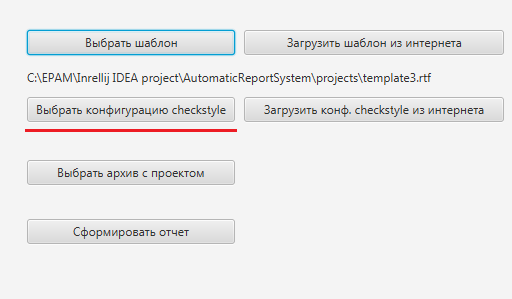


Рисунок 51 – Выбор файла с конфигурацией *checkstyle* вручную

На рисунке 52 представлено открывшееся диалоговое окно для выбора файла. Необходимо выбрать нужную конфигурацию и нажать кнопку «Открыть». Если было принято решение не загружать новый файл конфигурации, необходимо нажать кнопку «Отмена».

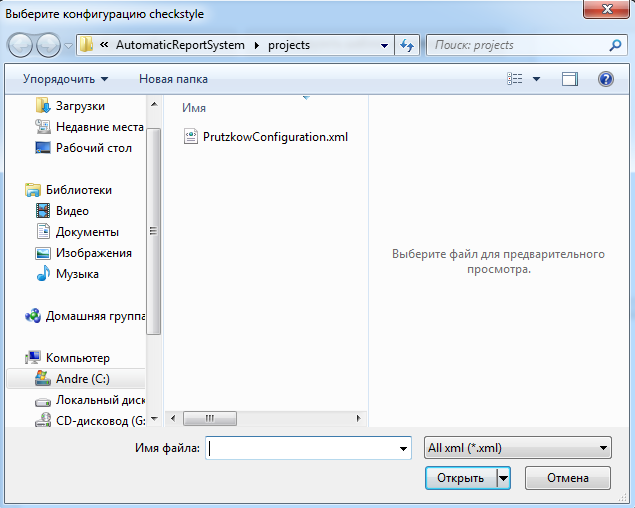


Рисунок 52 – Выбор конфигурации *checkstyle*

*Вариант 2.* Загрузка файла конфигурации *checkstyle* из интернета.

Нажмите на кнопку «Загрузить конф. *checkstyle* из интернета». Ее расположение представлено на рисунке 53.

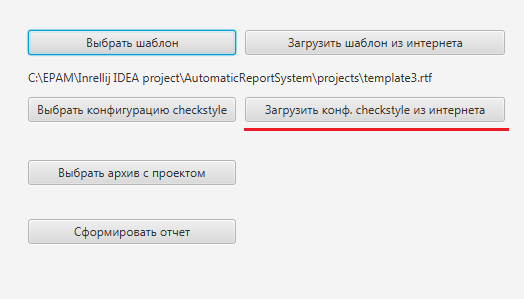


Рисунок 53 – Загрузка конфигурации *checkstyle* из интернета

На рисунке 54 представлено открывшееся диалоговое окно для загрузки файла конфигурации из интернета. Необходимо ввести в строку ввода *url* адрес и нажать кнопку «Загрузить конф. *checkstyle*». В случае если *url* будет неверным или окно ввода останется пустым, данное окно откроется еще раз. Если было принято решение не загружать шаблон из интернета, необходимо нажать кнопку «Отмена».

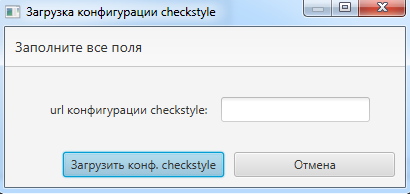


Рисунок 54 – Диалоговое окно ввода *url* для загрузки конфигурации *checkstyle* из интернета

На рисунке 55 представлен удачно загруженный шаблон. Под кнопками «Выбрать конфигурацию *checkstyle*» и «Загрузить конф. *checkstyle*» появился путь до файла с шаблоном.

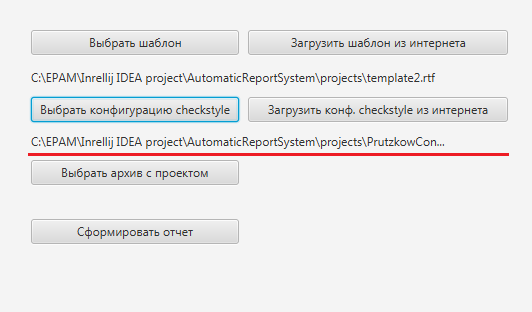


Рисунок 55 – Успешно загруженная конфигурация *checkstyle*

1. Загрузите архив с проектом. Для этого нажмите на кнопку «Выбрать архив с проектом». Ее расположение показано на рисунке 56.

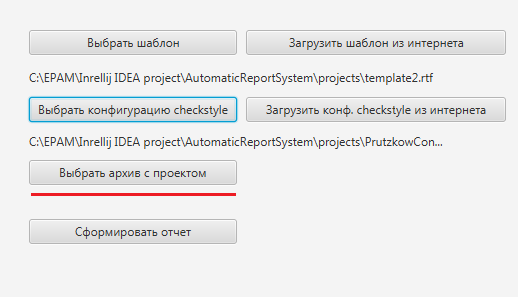


Рисунок 56 – Выбор архива с проектом

Если файл с шаблоном отчета не был выбран, то на экране появится ошибка. Она представлена на рисунке 57.

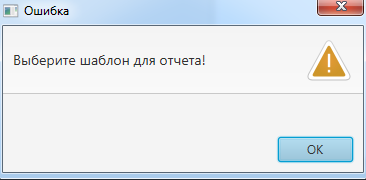


Рисунок 57 – Не выбран шаблон

Если файл с конфигурацией *checkstyle* не был выбран, то на экране появится ошибка. Она представлена на рисунке 58.

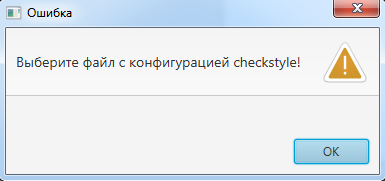


Рисунок 58 – Не выбрана конфигурация *checkstyle*

На рисунке 59 представлен удачно загруженный и проверенный архив с проектом. Под кнопкой «Выбрать архив с проектом» появился путь до выбранного архива. В правой части окна построилась иерархия файлов и папок. В нижней части окна появился результат проверки статическим анализатором кода *checkstyle*.

1. Выберите файлы, которые будут вставлены в листинг. Система предоставляет возможность выбора, как отдельных файлов, так и папки целиком. Если была выбрана папка, то все файлы и папки, которые находятся внутри нее, будут тоже выбраны. Пример выбора представлен на рисунке 60.

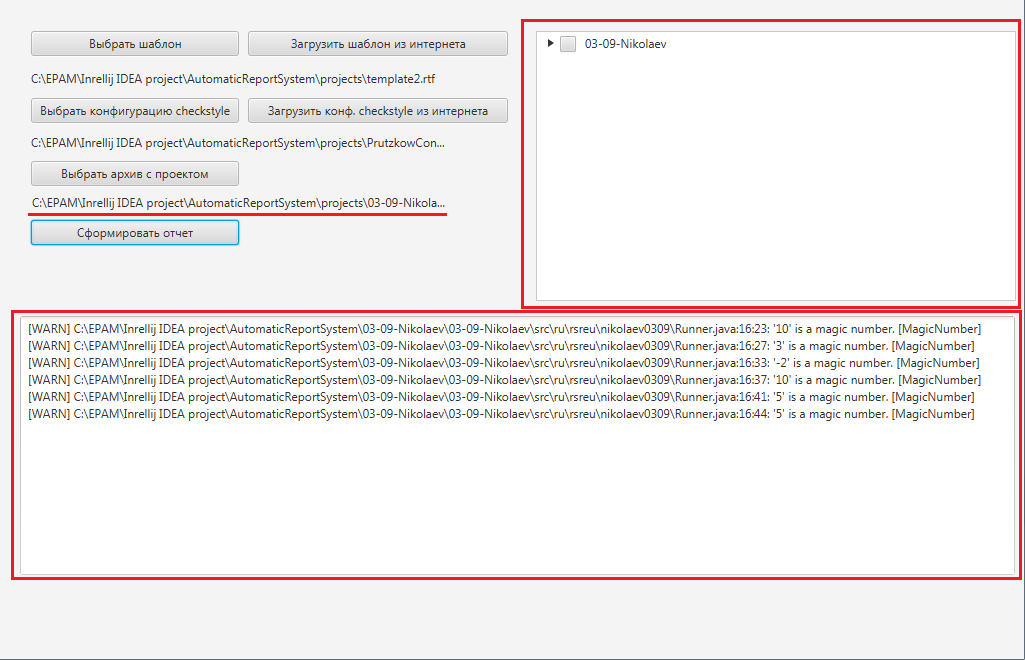


Рисунок 59 – Успешно выбранный архив с проектом

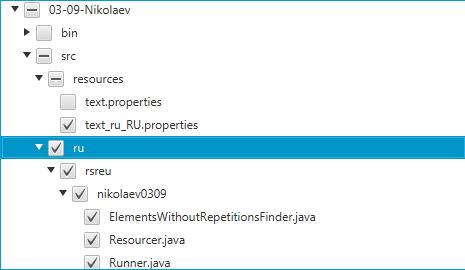


Рисунок 60 – Выбор файлов и папок для вставки в отчет

1. Сформируйте отчет. Для этого нажмите кнопку «Сформировать отчет». Ее расположение представлено на рисунке 61. На экране появится окно для ввода информации о пользователе. Данное окно изображено на рисунке 62.

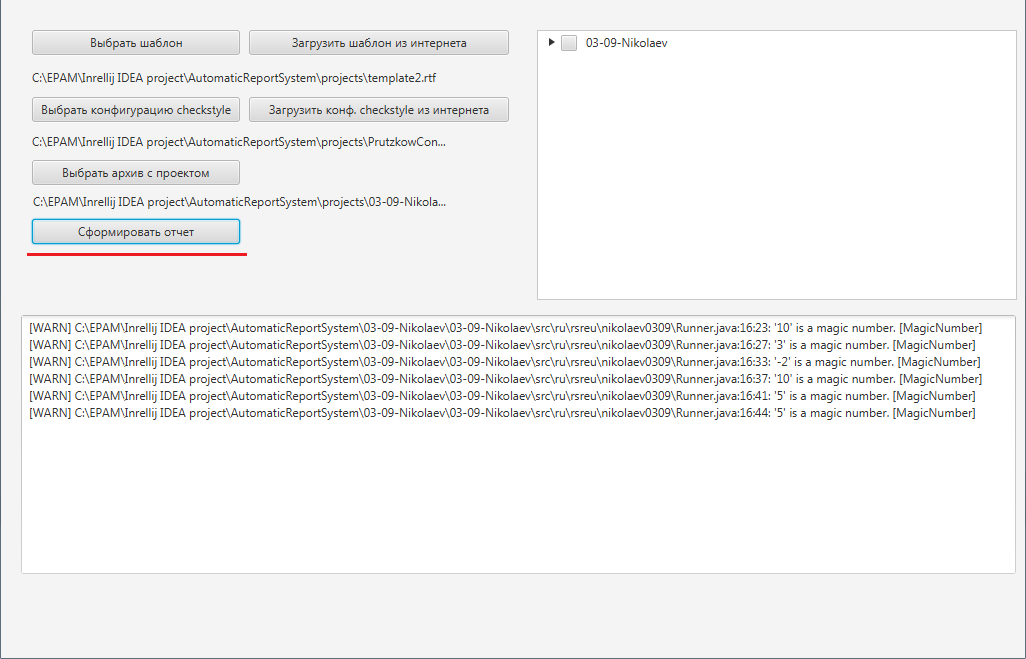


Рисунок 61 – Формирование отчета

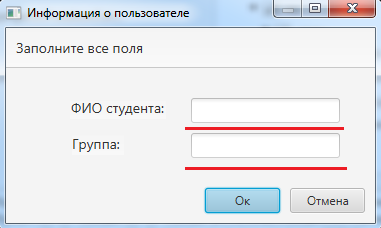


Рисунок 62 – Ввод дополнительной информации о пользователе

Если файлы для вставки в отчет были не выбраны, то отчет не сформируется, а на экране появится окно с сообщением об ошибке. Данное окно представлено на рисунке 63.

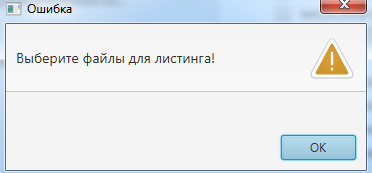


Рисунок 63 – Не выбраны файлы для вставки в отчет

В случае успешного формирования отчета, его можно увидеть в папке с приложением. Его название будет соответствовать названию архива, по которому этот отчет создавался.

Для проверки подписи отчетов необходимо указать системе папку с шаблонами, конфигурационными файлами *checkstyle*, архивами с проектами и отчетами. Процесс выбора происходит по аналогии с процессом ручного выбора шаблона и конфигурационного файла *checkstyle*. Окно с выбранными папками представлено на рисунке 64.

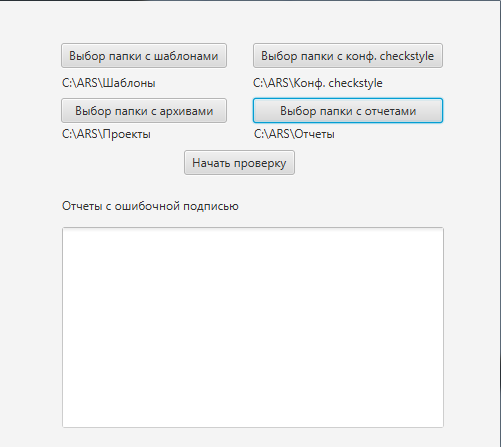


Рисунок 64 – Выбор папок для проверки подписи отчетов

После выбора папок необходимо нажать кнопку «Начать проверку». По окончании проверки выведутся все отчеты с ошибочной подписью. Результат проверки представлен на рисунке 65.

В результате разработки руководства пользователя рассмотрены способы формирования отчета и файла конфигурации *checkstyle*. Сформирована инструкция для пользования системой.

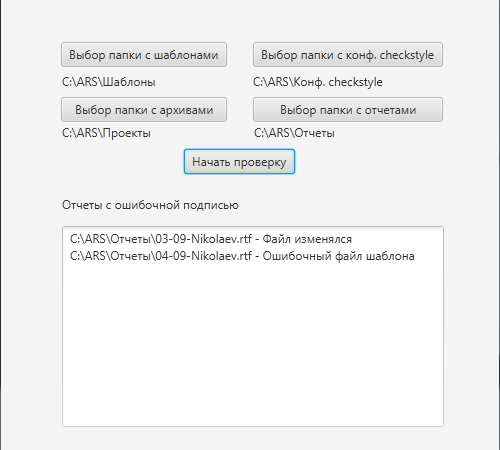


Рисунок 65 – Результат проверки подписи отчетов

# Тестирование

Тестирование является важной частью разработки программного обеспечения и проводится с использованием тест-кейсов. На этом этапе выявляются ошибки в работе системы, которые впоследствии должны быть устранены. Тесты должны покрывать весь основной функционал разрабатываемой программной системы. Протестирована основная функция системы: формирование отчетов. Файлы для шаблона и конфигурации *checkstyle* загружаются как вручную, так и с интернета.

Компьютер, на котором осуществлялось тестирование, имеет следующие характеристики:

* Процессор *AMD Athlon (tm) II X4 645 Processor 3.10 GHz;*
* Видеокарта *NVIDIA GeForce GTX 550 Ti;*
* Оперативная память 4.00 ГБ;
* Операционная система *Windows 7* домашняя расширенная 64-разрядная;
* Используемая версия *java jdk1.8.0\_144.*

Набор тест-кейсов называют тест-комплектом. Далее в таблице 15 представлен тест-комплект для системы формирования отчетов.

Таблица 15 – Тест-комлект для формирования отчета

|  |  |  |
| --- | --- | --- |
| **Описание теста** | **Ожидаемый результат** | **Результат теста** |
| **Запуск приложение**  1) Запустить приложение, кликнув лкм два раза по ярлыку. | Программа успешно запущенна. Появляется окно главного меню программной системы. | Пройден |
| **Загрузка шаблона:**  1) Запустить приложение, кликнув лкм два раза по ярлыку;  2) Нажать кнопку «Загрузить шаблон»;  3) Выбрать файл с шаблоном;  4) Нажать кнопку «Открыть». | Под кнопками загрузки шаблона появится путь до шаблона отчета | Пройден |

Продолжение таблицы 15

|  |  |  |
| --- | --- | --- |
| **Загрузка шаблона из интернета:**  1) Запустить приложение, кликнув лкм два раза по ярлыку;  2) Нажать кнопку «Загрузить шаблон из интернета»;  3) Ввести url;  4) Нажать кнопку «Загрузить шаблон». | Под кнопками загрузки шаблона появится путь до шаблона отчета | Пройден |
| **Загрузка конфигурации *checkstyle*:**  1) Запустить приложение, кликнув лкм два раза по ярлыку;  2) Нажать кнопку «Загрузить конфигурацию *checkstyle*»;  3) Выбрать файл с конфигурацией;  4) Нажать кнопку «Открыть». | Под кнопками загрузки конфигурации *checkstyle* появится путь до файла конфигурации | Пройден |
| **Загрузка шаблона из интернета:**  1) Запустить приложение, кликнув лкм два раза по ярлыку;  2) Нажать кнопку «Загрузить конф. checkstyle из интернета»;  3) Ввести url;  4) Нажать кнопку «Загрузить конф. *checkstyle*». | Под кнопками загрузки конфигурации *checkstyle* появится путь до файла конфигурации | Пройден |

Продолжение таблицы 15

|  |  |  |
| --- | --- | --- |
| **Загрузить архив с проектом:**  1) Запустить приложение, кликнув лкм два раза по ярлыку;  2) Выбрать файлы шаблона и конфигурации *checkstyle;*  3) Нажать кнопку “Загрузить проект”;  4) Выбрать файл с архивом;  5) Нажать кнопку «Открыть». | Под кнопкой загрузкой проекта появится путь до архива с проектом | Пройден |
| **Выбор файлов и папок для вставки в отчет:**  1) Запустить приложение, кликнув лкм два раза по ярлыку;  2) Выбрать файлы шаблона, конфигурации *checkstyle* и архива с проектом;.  3) Выберите в иерархии файлов и папок те файлы, содержимое которых необходимо вставить в отчет. | Выбранные файлы отмечены галочками | Пройден |
| **Создание отчета:**  1) Запустить приложение, кликнув лкм два раза по ярлыку;  2) Выбрать файлы шаблона, конфигурации *checkstyle* и архива с проектом;  3) Выбрать необходимые файлы и папки в окне с иерархией папок и файлов;  4)Нажать кнопку «Сформировать отчет»;  5)Ввести запрашиваемые программой данные о пользователе;  4) Нажать кнопку «Ок». | Отчет успешно сформирован и сохранен на компьютере в той же папке, что и программная система | Пройден |

В результате тестирования были выявлены все недостатки программной системы, в основном они связаны с ошибками, при неверном вводе пользователя. Все найденные ошибки устранены.

# ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы была разработана программная система автоматизации процессов проверки качества кода и формирования отчетов по практическим заданиям, которая имеет следующие функции:

- Автоматическое формирование отчета по практической работе в соответствии с шаблоном;

- Использование архива с исходным кодом в качестве входных данных для формирования отчета;

- Проведение проверки стиля и качества исходного кода в соответствии с определенными правилами;

- Защита отчета от последующих изменений.

Так же данная система является кроссплатформенным десктопным приложением. Данная система позволяет сократить время составления отчета с 10 минут до 1 минуты. Эти данные были получены в результате использования этой системы 13 студентами. Так же сократилось количество возвращенных на доработку практических заданий из-за ошибок в оформлении или проблем со стилем написания исходного кода. Эти ошибки исправляются студентами сразу без участия преподавателя.

Разработанная программная система будет внедрена в дисциплину «Программирование на *Java*». Также были подготовлены и отправлены документы для регистрации программного средства в РОСПАТЕНТ. Были опубликованы статьи в сборниках конференций НИТ-2017 [16] и СТНО-2018 [15].

В дальнейшем отдельные модули программной системы можно будет использовать в качестве плагинов для уже существующих систем, например *SonarQube*.

# Список использованных источников

1. Белладжио Д., Миллиган Т. Стратегия управления конфигурацией программного обеспечения с использованием *IBM Rational ClearCase.* – М.: ЛитРес, 2017. – 386 с.
2. Бенджамин Э., Мартин В. Java. Новое поколение разработки. – СПб.: Питер, 2013. – 560 с.
3. Блинов И.Н., Романчик В.С. Java. Методы программирования: учеб.-метод. пособие. – Минск: Изд-во «Четыре четверти», 2013. – 896 с.
4. Буч Г. и др. Введение в *UML* от создателей языка. – М.: ЛитРес, 2015. – 498 с.
5. Каюмова А.В. Визуальное моделирование систем в StarUML: Учебное пособие. Казань: Казанский федеральный университет, 2013. – 104с.
6. Кент Б. Экстремальное программирование: разработка через тестирование. – СПб.: Питер, 2017. – 224 с.
7. Крючков А. *Java.* Создание приложений. – М.: ЛитРес, 2018. – 359 с.
8. Куроуз Д., Росс К. Компьютерные сети. Нисходящий подход. – М.: ЛитРес, 2017. – 489 с.
9. Магазанник В. Человеко-компьютерное взаимодействие. – М.: Логос, 2017. – 256 с.
10. Макконнелл С. Совершенный код. Мастер-класс. – СПб.: БХВ-Петербург, 2017. – 896 c.
11. Мартин Р. К. Чистый код. Создание, анализ и рефакторинг. – СПб.: Питер, 2018. – 464 c.
12. Маршалл П., Тодд Б. Контекстная реклама, которая работает. Библия Google AdWords. – М.: Манн, Иванов и Фербер, 2014. – 464 с.
13. Миковски М., Поуэлл Дж. Разработка одностраничных веб-приложений. – М.: ЛитРес, 2017. – 514 с.
14. Москвитин А.А. Решение задач на компьютерах: часть II. Разработка программных средств: учебное пособие. – М.-Берлин: Директ-Медиа, 2015. – 427 c.
15. Николаев А.И. Проверка качества кода при автоматизированном составлении отчетов по практическим заданиям. [Текст]: Современные технологии в науке и образовании (СТНО-2018): сб. тр. XXI Всероссийской научно-техническая конференции студентов, молодых ученых и специалистов – Рязань: РГРТУ, 2018.
16. Николаев А.И. Программная система автоматизированного составления отчетов по лабораторным работам // Новые информационные технологии в научных исследованиях – НИТ-2017: материалы XXII Всероссийск. научн.-техн. конф. студентов, молодых ученых и специалистов./Рязан.гос.радиотехн.ун-т. – Рязань, 2017. – С. 184-185.
17. Нимейер П., Леук Д. Программирование на Java. – М.: ЛитРес, 2017. – 1214 с.
18. Орлов С.А. Теория и практика языков программирования. Учебник для вузов. 2-е изд. – СПб.: Питер, 2017. – 688 с.
19. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб.: Питер, 2016. – 366 с.
20. Роберт М. Чистая архитектура. Искусство разработки программного обеспечения. – СПб.: Питер, 2018. – 352 с.
21. Сильянов Д., Тараканов О. О ложных срабатываниях средств защиты информации. – М.: ЛитРес, 2017. – 12 с.
22. Скотт Ч., Бен Ш. *Git* для профессионального программиста. – СПб.: Питер, 2015. – 496 с.
23. Слива М. Кроссплатформенный подход как средство унификации обучения программированию в различных операционных системах – М.: ЛитРес, 2017. – 8 с.
24. Эберхард В. *Continuous delivery.* Практика непрерывных апдейтов. – СПб.: Питер, 2017. – 320 с.
25. *Baumeister H., Lichter H.* *Agile Processes in Software Engineering and Extreme Programming: 18th International Conference, XP 2017, Cologne, Germany, May 22-26, 2017, Proceedings. – New York: Springer*, 2017 – 306 c.
26. *Campbell A.G., Papapetrou P.P.* *SonarQube in Action. – New York: Manning Publications*, 2013 – 392 c.
27. *Google AdWords* [сайт]. – *URL:* [*https://adwords.google.com*](https://adwords.google.com)(дата обращения: 05.06.2017).
28. *Hall G. M.* *Adaptive Code: Agile coding with design patterns and SOLID principles, Edition 2. – London: Pearson plc*, 2017 – 448 c.
29. *Head First.* Паттерны проектирования. Обновленное юбилейное издание / Э. Фримен, Э. Робсон, К. Сиерра, Б. Берт. – СПб.: Питер, 2018. – 656 с.
30. *Krochmalski J**. IntelliJ IDEA Essentials. – Birmingham: Packt Publishing Ltd.,* 2014. – 276 c.
31. *Patel N., Patel K. Java 9* *Dependency Injection: Write loosely coupled code with Spring 5 and Guice. – Birmingham: Packt Publishing Ltd.,* 2018. – 246 c.
32. *UML @ Classroom:* *An Introduction to Object-Oriented Modeling / M. Seidl, M. Scholz C. Huemer, G. Kappel. – New York: Springer*, 2015 – 306 c.
33. *Venkat G.* *Rapid Modernization of Java Applications: Practical Business and Technical Solutions for Upgrading Your Enterprise Portfolio. – Delhi: McGraw Hill Education*, 2017. – 656 c.

# Приложение. Листинг наиболее значимых частей программы

*/\*\*  
 \* Тема ВКР: Разработка программной системы автоматизации составления отчетов по практическим заданиям  
 \* Выполнил: Николаев А.И. 09.03.04 гр.4413  
 \* Руководитель: Пруцков А.В., доктор технических наук, профессор каф. ВПМ  
 \* Средства разработки: Java, IntelliJ IDEA  
 \* Назанчение: Формирование отчетов и проверка качества кода  
 \* Дата разработки: 04.04.2018  
 \*/***public class** ARSController {

*/\*\*\*  
 \* Получение архива проекта  
 \** ***@param event*** *Событие формы  
 \*/*@FXML  
**private void** getFile(ActionEvent event) {  
 **if** (!checkTemplateAndConfigurationFileInputs()) {  
 **return**;  
 }  
 Node node = (Node) event.getSource();  
 FileChooser fileChooser = FileChooserConfigurationFactory.*makeProjectFileChooser*();  
 File file = fileChooser.showOpenDialog(node.getScene().getWindow());  
 **if** (file != **null**) {  
 **model**.setFile(file);  
 **labelFileName**.setText(file.getPath());  
 *//Generate tree* String unzipDirectory = **model**.unzipFile(file);  
 *//Checkstyle* **try** {  
 **checkstyleMessage**.setText(**model**.runChecks());  
 } **catch** (FileNotFoundException | CheckstyleException e) {  
 **checkstyleMessage**.setText(e.getMessage() + **"\n"** + e.getCause().getMessage());  
 }  
  
  
 **treeHandler** = **new** TreeHandler(**filesTreeView**, unzipDirectory);  
 **treeHandler**.createAllTree();  
 ARSModel.*deleteDirectory*(**new** File(**model**.*getUnzipDirectory*(file.getName())));  
 }  
}

*/\*\*\*  
 \* Выбор конфигурации  
 \** ***@param event*** *событие формы  
 \*/*@FXML  
**private void** chooseConfiguration(ActionEvent event) {  
 Node node = (Node) event.getSource();  
 FileChooser fileChooser = FileChooserConfigurationFactory.*makeCheckstyleConfigurationFileChooser*();  
 File file = fileChooser.showOpenDialog(node.getScene().getWindow());  
 **if** (file != **null**) {  
 **model**.setConfiguration(file);  
 **labelConfiguration**.setText(file.getPath());  
 }  
}

*/\*\*\*  
 \* Выбор шаблона для отчета  
 \** ***@param event*** *событие формы  
 \*/*@FXML  
**private void** chooseTemplate(ActionEvent event) {  
 Node node = (Node) event.getSource();  
 FileChooser fileChooser = FileChooserConfigurationFactory.*makeTemplateFileChooser*();  
 File file = fileChooser.showOpenDialog(node.getScene().getWindow());  
 **if** (file != **null**) {  
 **model**.setTemplateForReport(file);  
 **labelTemplate**.setText(file.getPath());  
 }  
}

*/\*\*\*  
 \* Формирование отчета  
 \*/*@FXML  
**private void** processFile() {  
 **if** (!checkAllFileInputs()) {  
 **return**;  
 }  
 Map<String, String> identifiersWithType = **null**;  
  
 **try** {  
 identifiersWithType = **model**.getAllIdentifiersFromTemplate();  
 System.***out***.println(identifiersWithType);  
 } **catch** (IOException e) {  
 AlertController.*showEmptyTemplateFileAlert*();  
 }  
 Map<String, String> identifiersWithText = **model**.fillIdentifiersWithText(identifiersWithType.keySet());  
 UserInformationController userInformationController = **new** UserInformationController(identifiersWithType, identifiersWithText);  
 Map<String, UserInformation> userInformation = **new** HashMap<>();  
 **while** (userInformation != **null** && !userInformationController.checkUserInformation(userInformation)) {  
 userInformation = userInformationController.show();  
 }  
 **if** (userInformation != **null**) {  
  
 List<File> selectedFiles = **treeHandler**.getAllSelected(**filesTreeView**);  
 **if** (!checkSelectedFiles(selectedFiles)) {  
 AlertController.*showNonSelectFileForListingAlert*();  
 **return**;  
 }  
 **model**.setFilesForListing(selectedFiles);  
 **model**.generateReport(**new** Report(userInformation, **checkstyleMessage**.getText()), **new** RtfReportWriter());  
 }  
}

*/\*\*\*  
 \* Проверка выбранных файлов в иерархии файлов и папок  
 \** ***@param selectedFiles*** *Выбиранные файлы  
 \** ***@return*** *Выбранные файлы прошли проверку или нет  
 \*/***private boolean** checkSelectedFiles(List<File> selectedFiles) {  
 **if** (selectedFiles.size() > 0) {  
 **return true**;  
 }  
 **return false**;  
}

*/\*\*\*  
 \* Проверка всех полей ввода  
 \** ***@return*** *Все поля заполнены верно или нет  
 \*/***private boolean** checkAllFileInputs() {  
 **boolean** result = **true**;  
 **if** (**labelTemplate**.getText().isEmpty()) {  
 AlertController.*showEmptyTemplateLinkAlert*();  
 result = **false**;  
 } **else if** (**labelConfiguration**.getText().isEmpty()) {  
 AlertController.*showEmptyCheckstyleConfigurationLinkAlert*();  
 result = **false**;  
 } **else if** (**labelFileName**.getText().isEmpty()) {  
 AlertController.*showEmptySourceFileLinkAlert*();  
 result = **false**;  
 }  
 **return** result;  
}

*/\*\*\*  
 \* Проверка загрузки шаблона и конфигурационного файла checkstyle  
 \** ***@return правильно ли загружены шаблон и конфигурационный файл*** *checkstyle**\*/***private boolean** checkTemplateAndConfigurationFileInputs() {  
 **boolean** result = **true**;  
 **if** (**labelTemplate**.getText().isEmpty()) {  
 AlertController.*showEmptyTemplateLinkAlert*();  
 result = **false**;  
 } **else if** (**labelConfiguration**.getText().isEmpty()) {  
 AlertController.*showEmptyCheckstyleConfigurationLinkAlert*();  
 result = **false**;  
 }  
 **return** result;  
}

*/\*\*\*  
 \* Класс контроллер для пользовательской информации  
 \*/***public class** UserInformationController {  
 **private** Map<String, String> **templateIdentifiersWithType**;  
 **private** Map<String, String> **templateIdentifiersWithText**;  
 **private** Map<String, UserInformation> **userInformationMap**;  
 **private** List<UserInformation> **userInformationList**;  
  
 **public** UserInformationController(Map<String, String> templateIdentifiersWithType, Map<String, String> templateIdentifiersWithText) {  
 **this**.**templateIdentifiersWithType** = templateIdentifiersWithType;  
 **this**.**templateIdentifiersWithText** = templateIdentifiersWithText;  
 }  
  
 */\*\*\*  
 \* Отображение диалогового окна для ввода пользовательской информации  
 \** ***@return*** *Пользовательская информация  
 \*/* **public** Map<String, UserInformation> show() {  
  
 *// Custom dialog* Dialog dialog = **new** Dialog<>();  
 dialog.setTitle(**"Информация о пользователе"**);  
 dialog.setHeaderText(**"Заполните все поля"**);  
 dialog.setResizable(**true**);  
  
 *//****TODO Переделать в map <Key, UserInformations>* if** (**userInformationMap** == **null** || **userInformationList** == **null**) {  
 **this**.**userInformationMap** = **new** HashMap<>();  
 **this**.**userInformationList** = **new** ArrayList<>();  
 **for** (Map.Entry<String, String> entry : **templateIdentifiersWithType**.entrySet()) {  
 **if** (TemplateIdentifier.*checkKeyTagOnUnresolved*(entry.getKey())) {  
 Label label = **new** Label(**templateIdentifiersWithText**.get(entry.getKey()));  
 TextField textField = **new** TextField();  
 UserInformation userInformation = **new** UserInformation(entry.getKey(), label, textField);  
 **userInformationList**.add(userInformation);  
 **userInformationMap**.put(entry.getKey(), userInformation);  
 textField.appendText(**userInformationMap**.get(entry.getKey()).getTextField().getText());  
 }  
 }  
 }  
  
  
 *// Создание и вывод окон ввода* GridPane grid = **new** GridPane();  
 grid.setAlignment(Pos.***CENTER***);  
 grid.setHgap(10);  
 grid.setVgap(10);  
 grid.setPadding(**new** Insets(20, 35, 20, 35));  
 **int** i = 1;  
 **for** (UserInformation userInformation : **userInformationList**) {  
 grid.add(userInformation.getLabel(), 1, i);  
 grid.add(userInformation.getTextField(), 2, i);  
 i++;  
 }  
 dialog.getDialogPane().setContent(grid);  
  
 *// Add button to dialog* ButtonType buttonTypeOk = **new** ButtonType(**"Ок"**, ButtonBar.ButtonData.***OK\_DONE***);  
 ButtonType buttonTypeCancel = **new** ButtonType(**"Отмена"**, ButtonBar.ButtonData.***CANCEL\_CLOSE***);  
 dialog.getDialogPane().getButtonTypes().add(buttonTypeOk);  
 dialog.getDialogPane().getButtonTypes().add(buttonTypeCancel);  
  
  
 *// Show dialog* Optional result = dialog.showAndWait();  
  
 ButtonType resultButton = (ButtonType) result.get();  
 **if** (!resultButton.getText().equals(**"Отмена"**)) {  
 **return userInformationMap**;  
 }  
 **return null**;  
 }  
  
 */\*\*\*  
 \* Проверка информации введенной пользователем  
 \** ***@param userInformation*** *Информация о пользователе  
 \** ***@return*** *Верно ли введена информация о пользователе  
 \*/* **public boolean** checkUserInformation(Map<String, UserInformation> userInformation) {  
 **boolean** result = **true**;  
 **if** (userInformation.size() == 0) {  
 result = **false**;  
 }  
 **for** (Map.Entry<String, UserInformation> entry : userInformation.entrySet()) {  
 **if** (TemplateIdentifier.*checkKeyTagOnUnresolved*(entry.getKey())) {  
 **if** (entry.getValue().getTextField().getText().trim().isEmpty()) {  
 result = **false**;  
 }  
 }  
 }  
 **return** result;  
 }  
}

*/\*\*\*  
 \* Аннотация для запуска методов анализа кода  
 \*/*@Target(ElementType.***METHOD***)  
@Retention(RetentionPolicy.***RUNTIME***)  
**public** @**interface** RunCheck {  
}

*/\*\*\*  
 \* Класс для анализа кода  
 \*/***public class** CodeChecker {  
 **private static** CodeChecker *instance*;  
  
 */\*\*\*  
 \* Реализация шаблона "Одиночка"  
 \** ***@return*** *Экзампляр данного класса  
 \*/* **public static** CodeChecker getInstance() {  
 **if** (*instance* == **null**) {  
 *instance* = **new** CodeChecker();  
 }  
 **return** *instance*;  
 }  
  
 */\*\*\*  
 \* Проверка кода с помощью checkstyle  
 \** ***@param file*** *Архив с проектом  
 \** ***@param configuration*** *Конфигурация checkstyle  
 \** ***@return*** *Результат проверки  
 \** ***@throws*** *FileNotFoundException Ошибка. Файл не найдет.  
 \** ***@throws*** *CheckstyleException Ошибка checkstyle.  
 \*/* @RunCheck  
 **public** CheckResult checkstyle(File file, File configuration) **throws** FileNotFoundException, CheckstyleException {  
 StringBuilder checkstyleResult = **new** StringBuilder();  
  
 List<ZipEntry> fileEntries = ZIPHandler.*getClassesEntry*(file);  
 **for** (ZipEntry entry : fileEntries) {  
 **if** (entry.getName().contains(**".java"**)) {  
 String sourceFilePath = ARSModel.*getUnzipDirectory*(file.getName()) + File.***separator*** + entry.getName();  
 checkstyleResult.append(Checkstyle.*start*(sourceFilePath, configuration.getAbsolutePath()));  
 }  
 }  
  
 **return new** CheckResult(checkstyleResult.toString());  
 }  
}

*/\*\*\*  
 \* Класс, где содержится основная бизнес-логика сситемы  
 \*/***public class** ARSModel {  
  
 */\*\*\*  
 \* Формирование отчета  
 \** ***@param report*** *Информация для отчета  
 \** ***@param reportWriter*** *Класс для сохранения отчета  
 \*/* **public void** generateReport(Report report, ReportWriter reportWriter) {  
 unzipFile(**file**);  
 report.setListing(ARSFileReader.*getStringFromFiles2*(**filesForListing**));  
 String hash = **null**;  
 **try**{  
 hash = Security.*getMD5Checksum*(**file**.getAbsolutePath(), **configuration**.getAbsolutePath());  
 } **catch** (Exception e){  
 e.printStackTrace();  
 }  
 reportWriter.writeReportFile(**templateForReport**,*getUnzipDirectory*(**file**.getName()) + **".rtf"**,report, hash);  
 *deleteDirectory*(**new** File(*getUnzipDirectory*(**file**.getName())));  
 }  
  
 */\*\*\*  
 \* Запуск проверок исходного кода  
 \** ***@return*** *Результат проверок  
 \** ***@throws*** *FileNotFoundException Ошибка. Файл не найден.  
 \** ***@throws*** *CheckstyleException Ошибка checkstyle.  
 \*/* **public** String runChecks() **throws** FileNotFoundException, CheckstyleException {  
 Class c = CodeChecker.**class**;  
 Method[] methods = c.getMethods();  
 StringBuilder stringBuilder = **new** StringBuilder();  
 **for**(Method mt : methods) {  
 **if** (mt.isAnnotationPresent(RunCheck.**class**)) {  
 *// Invoke method with appropriate arguments* **try** {  
 Object obj = mt.invoke(c.newInstance(),**file**, **configuration**);  
 CheckResult checkResult = (CheckResult) obj;  
 stringBuilder.append(checkResult.getResult());  
 } **catch** (IllegalAccessException | InvocationTargetException e) {  
 e.printStackTrace();  
 } **catch** (InstantiationException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 **return** stringBuilder.toString();  
 }  
  
 */\*\*\*  
 \* Удаление директории  
 \** ***@param directory*** *Удаляемая лиректория  
 \** ***@return*** *Результат удаления  
 \*/* **public static boolean** deleteDirectory(File directory) {  
 **if** (directory.exists()) {  
 File[] files = directory.listFiles();  
 **if** (**null** != files) {  
 **for** (File file1 : files) {  
 **if** (file1.isDirectory()) {  
 *deleteDirectory*(file1);  
 } **else** {  
 file1.delete();  
 }  
 }  
 }  
 }  
 **return** (directory.delete());  
 }  
  
 */\*\*\*  
 \* Получить все метки из шаблона  
 \** ***@return*** *Карта меток  
 \** ***@throws*** *IOException Ошибка чтения файла шаблона  
 \*/* **public** Map<String, String> getAllIdentifiersFromTemplate() **throws** IOException {  
 FileInputStream fio = **new** FileInputStream(**templateForReport**);  
 Map<String, String> map = Rtf.*template*(fio).findAllIdentificators();  
 fio.close();  
 **return** map;  
 }  
  
 */\*\*\*  
 \* Заполнить метки текстом  
 \** ***@param identifiers*** *Метки  
 \** ***@return*** *Карта меток  
 \*/* **public** Map<String, String> fillIdentifiersWithText(Set<String> identifiers) {  
 Map<String, String> identifiersWithText = **new** HashMap<>();  
 **for** (String identifierKey : identifiers) {  
 identifiersWithText.put(identifierKey, getIdentifierText(identifierKey));  
 }  
  
 **return** identifiersWithText;  
 }  
  
 */\*\*\*  
 \* Получение текста из метки  
 \** ***@param identifier*** *Метка  
 \** ***@return*** *Текст из метки  
 \*/* **private** String getIdentifierText(String identifier) {  
 **return** Resourcer.*getString*(identifier);  
 }  
  
 */\*\*\*  
 \* Разархивация файла  
 \** ***@param file*** *Архив  
 \** ***@return*** *Директория  
 \*/* **public** String unzipFile(File file) {  
 String unzipDirectory = *getUnzipDirectory*(file.getName());  
 ZIPHandler.*unZipIt*(file.getAbsolutePath(), unzipDirectory);  
 **return** unzipDirectory;  
 }  
  
 */\*\*\*  
 \* Получение директории для разархивации  
 \** ***@param fileName*** *архив  
 \** ***@return*** *Имя директории  
 \*/* **public static** String getUnzipDirectory(String fileName) {  
 String[] unzip = fileName.split(Pattern.*quote*(File.***separator***));  
 **return** unzip[unzip.**length** - 1].replace(**".zip"**, **""**);  
 }

}

*/\*\*\*  
 \* Класс для записи отчета  
 \*/***public class** RtfReportWriter **implements** ReportWriter {  
 @Override  
 **public void** writeReportFile(File templateForReport, String outputFileName, Report report, String hash) {  
 **try** (FileInputStream fio = **new** FileInputStream(templateForReport); FileOutputStream fos = **new** FileOutputStream(outputFileName)) {  
 Rtf.*template*(fio).info(RtfInfo.*hash*(hash)).injectAllNonReserved(report.getUserInformationMap())  
 .inject(**"Code"**, report.getCheckstyleResult() + report.getListing()).out(fos);  
 fio.close();  
 fos.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

*/\*\*\*  
 \* Класс для записи отчета в rtf файл  
 \*/***public class** RtfReportWriter **implements** ReportWriter {  
 */\*\*\*  
 \* Запись отчета в rtf Файл  
 \** ***@param templateForReport*** *Шаблон отчета  
 \** ***@param outputFileName*** *Имя сохраненного отчета  
 \** ***@param report*** *Информация для отчета  
 \** ***@param hash*** *Контрольная сумма  
 \*/* @Override  
 **public void** writeReportFile(File templateForReport, String outputFileName, Report report, String hash) {  
 **try** (FileInputStream fio = **new** FileInputStream(templateForReport); FileOutputStream fos = **new** FileOutputStream(outputFileName)) {  
 Rtf.*template*(fio).info(RtfInfo.*hash*(hash)).injectAllNonReserved(report.getUserInformationMap())  
 .inject(**"Code"**, report.getCheckstyleResult() + report.getListing()).out(fos);  
 fio.close();  
 fos.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

*/\*\*\*  
 \* Метки отчета  
 \*/***public class** TemplateIdentifier {  
 */\*\*\*  
 \* Значение меток отчета  
 \*/* Map<String, String> **templateIdentifierValue** = **new** HashMap<>();  
  
 **public** TemplateIdentifier() {  
 }  
  
 */\*\*\*  
 \* Проверка на допустимое название меток  
 \** ***@param keyTag*** *Тег метки  
 \** ***@return*** *Допустимое название или нет  
 \*/* **public static boolean** checkKeyTagOnUnresolved(String keyTag) {  
 **if** (!**"code"**.equals(keyTag.toLowerCase()) && !**"checkstyle"**.equals(keyTag.toLowerCase())) {  
 **return true**;  
 }  
 **return false**;  
 }  
  
 */\*\*\*  
 \* Добавить значени метка-текст  
 \** ***@param key*** *тег метки  
 \** ***@param value*** *текст  
 \*/* **public void** addValue(String key, String value) {  
 **templateIdentifierValue**.put(key, value);  
 }  
  
 */\*\*\*  
 \* Получить текст метки  
 \** ***@param key*** *тег метки  
 \** ***@return*** *Текст метки  
 \*/* **public** String getValue(Object key) {  
 **return templateIdentifierValue**.get(key);  
 }  
}

*/\*\*\*  
 \* Класс для работы с архивами  
 \*/***public class** ZIPHandler {  
 */\*\*\*  
 \* Получение информации для шаблона  
 \** ***@param file*** *шаблон  
 \** ***@param zipEntry*** *элементы архива  
 \** ***@return*** *Результат  
 \*/* **public static** String getDataForTemplate(File file, List<ZipEntry> zipEntry) {  
 ZipFile zf = **null**;  
 BufferedReader br = **null**;  
 StringBuilder sb = **new** StringBuilder();  
 **try** {  
 zf = **new** ZipFile(file);  
 **for** (ZipEntry entry : zipEntry) {  
 System.***out***.println(**"Read "** + entry.getName());  
 br = **new** BufferedReader(**new** InputStreamReader(zf.getInputStream(entry)));  
 String line;  
 **while** ((line = br.readLine()) != **null**) {  
 System.***out***.println(line);  
 sb.append(line);  
 sb.append(**"\n"**);  
 }  
 br.close();  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 } **finally** {  
 **try** {  
 **if** (br != **null**) {  
 br.close();  
 }  
 **if** (zf != **null**) {  
 zf.close();  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 **return** sb.toString();  
 }  
  
 */\*\*\*  
 \* Разархивация  
 \** ***@param zipFile*** *Архив  
 \** ***@param outputFolder*** *Конечная папка  
 \*/* **public static void** unZipIt(String zipFile, String outputFolder){  
  
 **byte**[] buffer = **new byte**[1024];  
  
 **try**{  
 *//create output directory is not exists* File folder = **new** File(outputFolder);  
 **if**(!folder.exists()){  
 folder.mkdir();  
 }  
  
 *//get the zip file content* ZipInputStream zis =  
 **new** ZipInputStream(**new** FileInputStream(zipFile));  
 *//get the zipped file list entry* ZipEntry ze = zis.getNextEntry();  
  
 **while**(ze!=**null**){  
  
 String fileName = ze.getName();  
 File newFile = **new** File(outputFolder + File.***separator*** + fileName);  
  
 System.***out***.println(**"file unzip : "**+ newFile.getAbsoluteFile());  
 *//create all non exists folders  
 //else you will hit FileNotFoundException for compressed folder* **new** File(newFile.getParent()).mkdirs();  
  
 FileOutputStream fos = **new** FileOutputStream(newFile);  
  
 **int** len;  
 **while** ((len = zis.read(buffer)) > 0) {  
 fos.write(buffer, 0, len);  
 }  
  
 fos.close();  
 ze = zis.getNextEntry();  
 }  
  
 zis.closeEntry();  
 zis.close();  
  
 System.***out***.println(**"Done"**);  
  
 }**catch**(IOException ex){  
 ex.printStackTrace();  
 }  
 }  
  
 */\*\*\*  
 \* Получение всех java классов из архива  
 \** ***@param file*** *Архив  
 \** ***@return*** *Список java классов  
 \*/* **public static** List<ZipEntry> getClassesEntry(File file) {  
 List<ZipEntry> zipEntries = **new** ArrayList<>();  
 **try** (ZipInputStream zin = **new** ZipInputStream(**new** FileInputStream(file))) {  
 ZipEntry entry;  
 String name;  
 **while** ((entry = zin.getNextEntry()) != **null**) {  
 **if** (entry.getName().endsWith(**".java"**)) {  
 zipEntries.add(entry);  
 name = entry.getName();  
 System.***out***.printf(**"Название: %s \n"**, name);  
 }  
 }  
 } **catch** (Exception e) {  
 System.***out***.println(e.getMessage());  
 }  
 **return** zipEntries;  
 }  
}

*/\*\*\*  
 \* Обработчик формы загрузки файлов  
 \*/***public class** DownloadFileController {  
 */\*\*\*  
 \* Диалоговое окно для загрузки файлов  
 \** ***@param labelName*** *сообщение для текстового поля  
 \** ***@param titleText*** *загооловок диалогового окна  
 \** ***@param buttonText*** *текст на кнопке  
 \** ***@return*** *текстовое поле  
 \*/* **public** TextField show(String labelName,String titleText,String buttonText) {  
 javafx.scene.control.Dialog dialog = **new** javafx.scene.control.Dialog<>();  
 dialog.setTitle(titleText);  
 dialog.setHeaderText(**"Заполните все поля"**);  
 dialog.setResizable(**true**);  
 *// Create layout and add to dialog* GridPane grid = **new** GridPane();  
 grid.setAlignment(Pos.***CENTER***);  
 grid.setHgap(10);  
 grid.setVgap(10);  
 grid.setPadding(**new** Insets(20, 35, 20, 35));  
 javafx.scene.control.Label label = **new** javafx.scene.control.Label();  
 label.setText(labelName);  
 javafx.scene.control.TextField textField = **new** javafx.scene.control.TextField();  
 grid.add(label, 1, 1);  
 grid.add(textField, 2, 1);  
 dialog.getDialogPane().setContent(grid);  
  
 *// Add button to dialog* ButtonType buttonTypeOk = **new** ButtonType(buttonText, ButtonBar.ButtonData.***OK\_DONE***);  
 ButtonType buttonTypeCancel = **new** ButtonType(**"Отмена"**, ButtonBar.ButtonData.***CANCEL\_CLOSE***);  
 dialog.getDialogPane().getButtonTypes().add(buttonTypeOk);  
 dialog.getDialogPane().getButtonTypes().add(buttonTypeCancel);  
  
  
 *// Show dialog* Optional result = dialog.showAndWait();  
  
 ButtonType resultButton = (ButtonType) result.get();  
 **if** (!resultButton.getText().equals(**"Отмена"**)) {  
 **return** textField;  
 }  
 **return null**;  
 }  
  
 */\*\*\*  
 \* Проверка текстового поля  
 \** ***@param textField*** *Текстовое поле  
 \** ***@return*** *Результат проверки  
 \*/* **public boolean** validate(TextField textField){  
 **if** (textField.getText().trim().equals(**""**)){  
 **return false**;  
 }  
 **return true**;  
 }  
}