ABERYSTWYTH UNIVERSITY

PROJECT REPORT FOR CS39440 MAJOR PROJECT

IntelliJ Plugin to Aid With Plagiarism Detection

Author:
Darren S. WHITE
(daw48@aber.ac.uk)

Supervisor:
Chris LOFTUS (cwl@aber.ac.uk)

April 29, 2018

Version: 1.0 (draft)

This report was submitted as partial fulfilment of a BSc degree in

Computer Science (G401)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, United Kingdom

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name: Darren S. White

Date: April 29, 2018

Consent to share this work

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Darren S. White

Date: April 29, 2018

Acknowledgements

I am grateful to...

I’d like to thank...

Abstract

Source code plagiarism is an ever-growing issue in academia, primarily in Computer Science. The majority of tools that exist to detect plagiarism only analyse the final piece of code. This paper discusses the research and development of a new tool which detects how the code was written. This tool will be an IntelliJ IDEA plugin and tracks file changes in the editor. The tracked data gives more of an insight into how plagiarism evolves over the course of development. This method of detection allows direct identification of the specific pieces of code that were plagiarised.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Overview

Plagiarism is becoming popular among students due to ease of access to the internet. Plagiarism.org states that, "One out of three high school students admitted that they used the Internet to plagiarize an assignment" [1]. Detecting acts of plagiarism is not a simple task, especially when done manually. Humans are simply not capable of analysing multiple pieces of work and finding similarities. At least not efficiently or quickly. Automatic detection systems already exist but they only analyse the final piece of work (these systems are described more in detail in section 2.1). These systems can be improved by advancing the detection algorithms. This may prove difficult over time as these algorithms become more complex.

A system that analyses work from its inception would allow more targeted and sophisticated detection methods to be performed. These methods could directly identify the plagiarised work, and how it may be transformed to hide any evidence. This system would not act as a sole detection system, but instead alongside existing tools. This would improve the detection rate and provide more evidence for such cases.

1.2 Plagiarism and Unacceptable Academic Practice

Aberystwyth University classifies plagiarism, collusion, and fabrication of evidence of data as acts of UAP (Unacceptable Academic Practice) [2]. The act of plagiarising is to commit fraud by stealing someone's work and not clearly referencing the owner [3]. Plagiarism has many forms, some of these are described below [2] [3] [4] [5].

- Copying or cloning another's work without modification (including copying and pasting)
- Paraphrasing or modifying another's work without due acknowledgement
- Using a quotation with incorrect information about the source
- The majority of the work is made up from other sources
- Copying an original idea from another persons work

- Putting your own name on someone else's work

These forms of plagiarism apply to written work as well as source code. In terms of coding, copying someone's code without modification or acknowledgement is still plagiarism. Automated systems can be put in place to detect plagiarism and UAP.

The main objective of detecting plagiarism, is finding pieces of work which originated from another source. The plagiarised work can often be obfuscated or modified in a way to try and conceal the original work yet keep the same outcome. In source code, this ranges from simple changes to more complex alterations. The program plagiarism spectrum describes the levels of plagiarism that can be done, with level 0 having no modifications and level 6 altering the control flow of the program [6]. The higher levels make it more difficult to compare pieces of work against each other. Tools that only analyse the final piece of work will struggle with identifying the higher levels in the spectrum. Tracking how a piece of work develops over time will show patterns of obfuscation. As an example, evidence of changing control flow will be tracked. This shows us that the potential of this project could potentially be very successful if developed properly.

Chapter 2

Background

This project introduces two problems. Tracking the work as it is being written, and identifying when plagiarism has occurred. Tracking code being written will be accomplished by developing a plugin for IntelliJ IDEA. IntelliJ IDEA is a Java IDE (Integrated Development Environment) for software development. IntelliJ has to capability to add and develop plugins. These plugins are developed using the IntelliJ Platform [7]. Eclipse Identifying plagiarism is the second major task for this project. Due to this system operating differently to existing systems, it will be difficult to determine how to accurately detect plagiarism. Alternatively, Eclipse could have been used to develop the plugin. Eclipse is also a Java IDE for developing Java software. Similar to IntelliJ, Eclipse has the ability to add and develop plugins.

"Plagiarism detection usually is based on comparison of two or more documents" [8]. This is what makes this project stand out for me. It's not simply reinventing the wheel, but finding new ways to solve problems which still exist in academia.

2.1 Existing Systems

The most popular existing systems are Turnitin, MOSS, JPlag, and YAP3, amongst many others. For this system, MOSS and JPlag are worth looking into as they both check source code, whereas Turnitin only checks plain text [8]. Although these tools can automatically identify plagiarism, they still require human verification.

MOSS relies on the Winnowing detection algorithm. It works by creating fingerprints or hashes for documents [9]. Creating single hashes for each document allows for exact document comparisons. Single hashes are useful for checking if a document is correct and non-corrupt. Instead, Winnowing, uses multiple k-grams for each document. K-grams all for partial document comparisons between multiple documents. Comparing between multiple sources does however require a large set of documents beforehand.

JPlag provides an online user interface where documents can be submitted and the results can be viewed. JPlag parses each document into token strings and these tokens are compared in pairs between documents. The percentage of tokens that match is referred to as the similarity factor [10].

Yap3 operates in a similar way to JPlag and MOSS. It uses its own similarity detection algorithm, RKR-GST. The algorithm compares sets of strings in a text much like the other

algorithms [11].

Plagiarism detection tools can use either extrinsic or intrinsic detection algorithms. Extrinsic detection uses external sources to compare against. Using a massive collection of external documents, extrinsic algorithms can be used very effectively although will take a large amount of time to process. Intrinsic detection analyses the document to detect changes in writing style. This allows the post-processing time to be very small in comparison to extrinsic algorithms. All of the existing systems described above use extrinsic detection algorithms.

2.2 IntelliJ Plugin SDK

IntelliJ is being used as it was a supervisor requirement. Being unfamiliar with the IntelliJ Plugin SDK, I delved deep into the online tutorials provided by JetBrains [12]. IntelliJ comes bundled with the IntelliJ Platform Plugin SDK so setting up the development environment should be no issue. The IntelliJ Community Edition is open source and contains many plugins [13]. This repository is very useful. Looking at existing plugins is sometimes more useful than reading tutorials.

One aspect of the plugin that would be needed is to track keyboard events. I set out to try and find what possible methods there were of doing this. This feature is not mentioned in the tutorial, so I went digging through the SDK in the Community Edition repository. TypedHandlerDelegate and TypedHandler are classes used to perform actions upon typing events in the editor of the IDE. Both of these classes could be very useful when starting development.

Saving data to disk is also another feature that would be used by the plugin. This feature is used widely by many plugins and was documented in the tutorial. Persistent state is stored to file using the XML format. The tutorial was understandable but I decided to take a look at an existing plugin for a real example, the GitHub plugin. GitHub saves settings to file and so this was helpful to my understanding.

2.3 Back-end Server

The following is a list of the possible technologies that could be used for the back-end server. The requirements for the back-end server include being able to submit tracked data and storing it in a suitable database. Authentication will also be required. A front-end service will be used to display information about the tracked data to authenticated users. Having experience with both Python (Flask) and Ruby on Rails to develop web servers proposes a major decision. Bootstrap and jQuery can both supplement the front-end user experience.

- **Python / Flask** - Micro web framework for Python based on Werkzeug, and Jinja2.
- **Ruby on Rails** - Server-side web framework written in Ruby which uses a MVC architecture and provides default structures. It is very quick to implement a solution.
- **Bootstrap / JQuery** - Bootstrap is a front-end library for HTML, CSS, and JavaScript. JQuery is a JavaScript library.
- **Database** - Either an SQL or NoSQL database will be used to store recorded data for users.

- **Authentication** - Staff (and possibly students) should be able to sign-in to the service

The server could work in various ways. Recorded data could be sent continuously or once. It could receive continuous data from the plugin which would allow the user to seamlessly use the plugin without having to interact with the server in anyway. This would require the user to input some identification in the plugin.

Another way which the server could work is that the user could submit the recorded data in a form. This would reduce the bandwidth used by the server. But this introduces the issue that the user could modify the saved data before submitting. The would also make it easier to switch between projects and different computers.

A front-end application will also be required for staff and potentially students. Staff should be able to view students project submissions and the final plagiarism result. Both Rails and Flask support this and can be tied into the back-end module.

2.3.1 Post-processor

The post-processor would be a part of the server. Its functionality would be to process the data recorded from the plugin. The resulting data would be stored in the database, which the server can display when requested. There is also the possibility to add machine learning to the post-processor to improve detection results.

2.4 Analysis

The project will include the IntelliJ plugin, back-end server, post-processor, and a database. The plugin will be developed using the IntelliJ Plugin SDK using IntelliJ IDEA. The plugin will handle tracking file changes in the editor of the IDE. The tracked changes will be stored in the back-end server database. Each students' project will be stored with identification data. Identification data should include Aberystwyth user id, full name, project title, and the project module.

The back-end server will be developed using Python 3 and Flask. I decided to use Python due to being more confident and experienced with Python and Flask than Ruby on Rails. The back-end server will operate either continuously or non-continuously. This decision is discussed more in detail in Iteration 1 in chapter 4. Both back-end server scenarios are described below in subsection 2.4.1 and subsection 2.4.2. The server should also provide authentication for staff and potentially students. It would be nice to have authentication using Aberystwyth credentials. The post-processor will also be developed using Python 3 for consistency. The job of the post-processor will be to process all the data and store the results in the database. MongoDB will be used for the database. MongoDB stores documents in BSON (Binary JSON) format. Due to the recorded data being stored as XML, it is convenient to convert to a JSON format. Complex queries are not required, only simple queries are needed to store, retrieve, and update student submissions.

2.4.1 Continuous back-end server

The student will only need to interact with the plugin in this scenario. The user will be required to enter the identification data upon project creation. This identity data will be sent to the server. All of the tracked data will be sent to the server along with some of the identity data to be stored in the database. To prevent malicious attacks, the connection between the plugin and the server should be encrypted. Once the project is complete the student will click an action which will notify the server of the completion to start post-processing of the data. The front-end application will allow staff to login and view their students submissions as well as the results from post-processor. This design does require a constant connection to the server, so a fail-safe would be needed for when no network is connected

2.4.2 Non-continuous back-end server

In this scenario, the user will interact with both the plugin and the front-end application. There are no prerequisites to setting up the plugin. Instead the plugin tracks file changes of the editor and saves them to an XML file. Once the student has finished their project, they will submit the XML file. This does present an authenticity issue. Having this stored to a plain XML file, means that the student may maliciously manipulate the data. This would provide the server with spurious data providing an incorrect plagiarism result. This problem could be solved by encrypting the file. Students should be able view their previously submitted projects. Staff however, should be able to view their students project submissions and the results from the post-processor. This design does not require a constant network connection.

2.4.3 Requirements

A list of key requirements are shown below. Using an agile process, these are the initial objectives identified. These may change in future sprints. Each item will be split into a single or multiple stories depending on the size of the task. More on the agile process used is described in section 2.5

- **Design architecture decisions**
  - **Data structure for storing data**
    The data that is recorded must be stored in an adequate and efficient data structure. A Tree or Map implementation may suffice.
  - **Data submission method**
    Recorded data could be sent to the server continuously (real-time) or non-continuously.
- **Development**
  - **IntelliJ Platform Plugin**
    - Implementation of source code detection methods:
      - Keyboard events
      - Copy/paste
      - Code generation
      - Code auto-completion

        - Refactoring
        - External changes (using a different editor)
    - Settings GUI - for student identification information (only required for continuous server)
    - Storing recorded data in a data structure
    - Storing recorded data to file (only required for non-continuous server)
    - Encrypt stored data (only required for non-continuous server)
    - Unit tests
- **Back-end server**
    - Database storage
    - Authentication for staff (and potentially students)
    - Docker support for quick deployment
    - Mechanism to submit recorded student data
    - Unit tests
- **Post-processor**
    - Watch for updates in the database - this would allow the post-processor to process new student submissions
    - Process recorded data and update the database with the result
    - Unit tests
- **Front-end application**
    - A web application for staff to use (and possibly students)
    - Authentication page for staff (and possibly students)
    - Display student project submission data
    - Staff should be able to view all students' submission data
    - Students should only view their own submissions
    - Possible cases of plagiarism should be shown with adequate evidence. Graphs, metrics, and tracked data could be displayed.

2.5 Process

The approach chosen to use for this project was an agile methodology, scrum. Scrum was chosen to due having prior experience with this methodology. Scrum usually consists of multiple team members. This project was developed individually and this meant that a modified scrum was used. Scrum uses short iterations, each consisting of planning at the beginning, implementation, review, and then retrospective at the end. Alongside these iterations, short daily stand-up meetings are attended by the team. Daily stand-up is a short 10 or 15 minute meeting where the team discusses what they're working on and of any changes that need to be reviewed. Instead of daily stand-up, weekly meetings on Mondays with my supervisor were organised. These meetings consisted mostly of discussions of the previous and next sprints.

Planning involves discussing and deciding which stories should be worked on during the sprint. A story is a piece of work that needs to be done. The intricate details of each story may not yet be known but they will develop over the course of the iteration. A story will have a time estimate associated with it. The golden ratio is used as a guideline, and the story points are described below.

- **1** - 10 minutes to 1 hour
- **2** - A few hours to half a day
- **3** - A few days
- **5** - A week
- **8** - Over a week, this story should be broken into smaller stories

Implementation and review take up most of the iteration time. This is spent designing, developing, and reviewing code that will end up in the code base. Once code has been reviewed for a story, it can be marked as done.

Retrospective is a reflective process. It is a discussion of what went well, what didn't go well, and what could change for the next sprint. The retrospective is aimed to improve the scrum process over time.

To track each sprint and its stories, I used milestones and issues on GitHub [14]. During planning I would create a new milestone, assign issues to it (creating new issues if necessary), and set a goal. The goal would be a general aim for that sprint, which multiple stories would accomplish. During implementation and review, issues can easily be closed by referencing them in a commit message with specific keywords such as `Fixes #IssueNum` [15]. After the sprint is done, I would close the milestone. Any remaining issues in the milestone would remain in the backlog still marked as open.

The structure of this report will continue with each iteration as a new chapter. The final design will be discussed in detail in chapter 12. The testing approaches used for each of the project components will be discussed in chapter 13.

Chapter 3

Iteration 0

3.1 Planning

This first iteration involved investigating the IntelliJ Plugin SDK and starting the initial development of the plugin. Below is the list of stories and their assigned story points. The list is displayed in completion order. Bugs that were encountered and resolved during this iteration are also included. Bugs do not have story points as they are not planned stories.

- Investigate IntelliJ Platform Plugin SDK component: 2 points
- Research existing plagiarism detection tools: 3 points
- Implement Settings GUI: 2 points
- Investigate data structure for storing data: 1 point
- Detect copy-paste of code: 2 points
- Store recorded data in a data structure: 2 points
- Detect external file changes: 3 points
- Bug: File path is sometimes relative and has no parent
- Bug: Ignore .idea directory when listening for DocumentEvents
- Bug: FileTooBigException when comparing external file changes
- Bug: Check externally changed file exists

3.2 Implementation

3.2.1 Initial Investigation of IntelliJ Plugin SDK

The *Getting Started* tutorial from JetBrains is an obvious place to start [12]. Setting up a new project with the IntelliJ Plugin SDK was very simple, as the SDK is bundled with IntelliJ IDEA.

Following the tutorial to add actions to the plugin was straight-forward. `AnAction` is a class which can be subclassed to perform an action when an a menu item or toolbar button is clicked. `AnAction` is registered to a menu item or toolbar button in the `plugin.xml` file.

The `plugin.xml` file is used to register actions and project components. These components can be one of three levels: application, project, or module [16]. An application level component is created and initialised upon the start-up of the IDE. A project level component is created for each project in the IDE. A module level component is created for each module inside of each project in the IDE.

`AnAction` has been added to a new menu. The new menu is specifically for this plugins actions. The action will display an input dialog when clicked. This will allow the student to enter their Aberystwyth username. When the action is clicked again, an information dialog will display the previously entered username.

`PersistentStateComponent` is a class used to store data to a file. Using this in conjunction with the previously used `AnAction` implementation allows saving the students' username to file and loading it when the project is loaded. This XML file is saved in the .idea directory.

The TypedHandler and TypedHandlerDelegate classes can both be used to listen to typing events in the editor. Implementing the TypedHandlerDelegate was a simplistic solution. Although, as with anything simple, there is always a drawback. That drawback is when the auto-complete popup is shown, the typing events are no longer trigged. This means that most of the typing will not be tracked. On the other hand, implementing a TypedHandler class to override the default class worked well but also came with a dramatic downside. The auto-complete popup no longer worked at all. There had to be another way to track typing events in the editor. Introducing, `DocumentListener`, this class listens to file changes in the editor. This means that any change to a files contents will trigger an event. This works perfectly, even with the auto-complete popup. The `DocumentEvent` object contains all of the data that describes what was changed in a file.

3.2.2 Developing the Initial Data Structure

The `Change` class is the core of the data structure implementation (see Figure 3.1 below for the UML Class Diagram). Whenever a file is modified a new `Change` object is created to represent the data that was altered. Initially, a LinkedList of Changes was used to store all recorded data. A LinkedList was chosen due to keeping insertion order, but it wasn't fully necessary as each Change had an associated timestamp. However, the LinkedList was superseded by the HashMap. The key was the file path, and the value was a FileTracker object (see Figure 3.2 below for the UML Class Diagram). The FileTracker now contains the LinkedList of Changes object. The final data structure design and example XML data is discussed more in chapter 12.

Change
+ offset: int + oldString: String + newString: String + source: Source + timestamp: long
+ Change() + Change(int,String,String,Source,long)

Figure 3.1: The UML Class Diagram for the `Change` class. Each of the fields are public and non-final to allow for serialisation which is required for storing state via `PersistentStateComponent`. The default constructor is also needed for serialisation. Initially the this class also had a path:String field but was removed when moving to a Map data structure.

FileTracker
+ path: String + changes: LinkedList<Change> + cache: String
+ FileTracker() + FileTracker(String)

Figure 3.2: The UML Class Diagram for the `FileTracker` class. Each of the fields are public and non-final to allow for serialisation which is required for storing state via `PersistentStateComponent`. The default constructor is also needed for serialisation.

3.2.3 Identifying Change Sources

Now that file changes can be detected and stored in an adequate data structure, the source of the changes must be identified. Below is a list of identifiable sources. The first source detection that was implemented was clipboard (i.e. copying and pasting). The `CopyPasteManager` class contains methods to retrieve the current clipboard contents. To identify if a change originated from the clipboard, the change `newString` value must be compared with the clipboard contents. IntelliJ IDEA supports clipboard history so each content value must be checked. If the values match then the change must have been a copy-paste event. Currently, any unidentified change will be classed as having a source of `other`. This includes normal typing. The reasoning behind this is to identify all types of source changes and then the remaining unidentified changes would only be from the student typing. But for now, they will be classed as `other`.

Identifiable Sources:

- Keyboard events
- Copy/paste
- Code generation

- Code auto-completion
- Refactoring
- External changes (using a different editor)

The last story for this sprint is to implement detecting external file changes. External file changes occur when the project is closed. When using an external editor (i.e. any other editor besides IntelliJ IDEA) to edit any of the project files, these should be detected when the project is next opened. The `FileTracker` class has a String `cache`. This will be used to store the last known file contents for that file (encoded as base-64). Each time a new change is added to a `FileTracker` its cache is also updated. This ensures the `cache` is always up-to-date. Upon opening the project, the `cache` of each `FileTracker` is compared against its current file contents. Unfortunately, there wasn't enough time in this sprint to finish implement an algorithm to determine the differences between the old and new file contents.

3.2.4 Squashing the Bugs

Multiple minor bugs were encountered near the end of this iteration. The first was a simple `NullPointerException` when adding a new change for a file. If the file had no parent file, then getting its path would throw an `NullPointerException`. The next issue was less of a bug, and more of an inconvenience. Files in the `.idea` directory were being tracked. This was causing the XML file containing the tracked data to track itself and this caused the a massive increase in the file size very quickly. The last two bugs were associated with the new external file change detection. When reading the new contents of a large file, a `FileTooBigException` was being thrown. This was solved by directly reading from the file `InputStream`. Lastly, externally changed files were not checked if they exists (i.e. they were deleted when the project was closed). Originally, the removed file was to be removed from the tracked list. But, it was decided to add a change which removed all the content instead.

3.3 Retrospective

This first sprint went exceptionally well considering the unfamiliarity with the IntelliJ Plugin SDK. Already having three different methods of detecting file changes in the editor is daunting but the last method definitely looks like the best option. The velocity for this sprint is 15. This is quite high, so the story estimations could be too large. The next sprint will take this into consideration when estimating story points.

Chapter 4

Iteration 1

4.1 Planning

This iteration revolved around setting up the back-end server and adding basic unit tests to the plugin. Aberystwyth authentication for staff (and potentially students) will be a major stepping stone for this sprint. Below is the list of stories and their assigned story points. The list is displayed in completion order.

- Add external change algorithm to find the difference: 2 points
- Investigate testing framework for IntelliJ Plugin: 2 points
- Write a test for copy-paste detection: 1 point
- Write external change detection test: 2 points
- Investigate server authentication: 3 points
- Investigate data submission method: 2 points
- Investigate back-end server software: 3 points
- Docker support for quick deployment: 2 points
- Implement web server: 1 point
- Add database storage: 2 points
- Implement server authentication: 2 points
- Check if user is staff or student: 2 points
- Add dashboard bootstrap template: 1 point

4.2 Implementation

4.2.1 Implementing the External Change Detection Algorithm

At the end of the last sprint, the external file change detection was implemented in the plugin. This needed an algorithm implemented to check the differences between the old and new file content. This requires a complex algorithm. After researching, the *Myers' diff algorithm* was exactly what was needed. Originally, a third-party library implementation was going to be used, but IntelliJ already includes difference comparisons in the IDE. The SDK provides this functionality too. Albeit difficult at first, due to minimal documentation or examples provided. The `ComparisonManagerImpl` class provides methods to compare the characters between two strings and returns a list of `DiffFragments`. After converting these to a list of `Changes`, they were added to the appropriate `FileTracker` instance.

4.2.2 Testing the Plugin

The options for adding units tests to the plugin were limited to one choice. Due to the plugin being written in Java, JUnit was chosen. The IntelliJ Plugin SDK also provides an API for testing. The testing infrastructure provided by the SDK allows running tests in a headless environment. This means that the plugin can be tested without the UI while keeping the full functionality of the IDE. Despite the absurdly long class name, `LightPlatformCodeInsightFixtureTestCase`, provides the basics for unit tests with the IntelliJ Plugin SDK. Extending from `LightPlatformCodeInsightFixtureTestCase`, the `BaseTest` class was built to contain all of the useful testing methods for the plugin. This included methods such as `assertChangeListSize` and `assertOneChangeMatches`. The former method tests that the list of changes for a given file path is of a certain size. The latter method tests that the list of changes for a given file path contains a matching `Change` which is validated by the `Predicate`.

Extending from the `BaseTest` class, a test class for copy-paste detection was to be developed. This class contains one method which tests that when the clipboard is used to paste contents into the editor, it is identified properly. The testing infrastructure allows a file to be created in the test project, then a string value can be added to the clipboard and the editor can paste the clipboard contents. This data gets tracked as usual and then the file changes are tested against to verify that it was identified from the clipboard.

The second testing class added was to test the external file change detection. This is more complex than the simple copy and paste detection test. The project needs to be closed and the file must then be changed. This proved to be quite the challenge. To achieve this, originally, the project was actually closed but this prevented any files to be editable in the project. Instead, the `ProjectDocumentListener` which tracks all the file changes, was notified of the project closing, but the project would not be closed. Upon the project closing, the listener is removed from the project and file changes are not longer tracked. This allowed files to be editable, and in turn, would mimic external file changes. The `ProjectDocumentListener` was then notified that the project is opened and so the external changes would be detected and verified.

4.2.3 Adding Authentication to the Server

Prior to the development of the back-end server, the authentication mechanism was to be investigated. A member of IMPACS (Institute of Mathematics, Physics and Computer Science) was contacted about authentication via Aberystwyth University. This would allows staff and students to authenticate with the back-end server using their Aberystwyth credentials. This would minimise the amount of development needed for an independent authentication system. It would also remove the need for users to register with the new system. Instead, utilising the Aberystwyth LDAP (Lightweight Directory Access Protocol) server, users can be authenticated using their Aberystwyth University credentials. After the LDAP3 server details were acquired, this could be tested and verified. The back-end server is to be written in Python 3. The LDAP3 library provides the necessary API to connect to the Aberystwyth University LDAP server [17]. Following the documentation, a simple authentication method was developed to test the authentication functionality. Using the LDAP3 library, an LDAP server was created with a new connection for each user. Attempting to `bind` to connection would result in either success or failure. If the bind was successful, then the user is authenticated and further user information may be retrieved from the connection.

4.2.4 Deciding on the Server Submission Method

Before continuing development on the back-end server, the data submission method must be decided. This is the choice of either the continuous or non-continuous back-end server. The Analysis contains details on both of these methods. In the end, the non-continuous server was chosen. This is due to the simplicity of the design. It provides better support for offline development, and students will not be required to interact with the plugin. However, the server will have to provide authentication for both staff and students, as well as separate user interfaces for both. Students will also have to submit their tracked data via the user interface but this should be less complex than continuously sending the data through a network protocol.

4.2.5 Parsing and Storing the XML Data

Now that the chosen authentication method is implemented and the data submission method has been chosen, the back-end server development can continue. The back-end server will need to parse the tracked XML data file. The database being used to store the tracked data is MongoDB which is a NoSQL database. The parsed XML data will need to be transformed into a JSON-like document. This didn't prove difficult due to the similarities between the structure of JSON and XML.

Using the `xml.etree.ElementTree` module, the XML data was parsed. But due to the format that the `PersistentStateComponent`, the XML parsing needed a little more intelligence. The XML data contains nested lists in a map, so these data structures would need to be correctly parsed. Recursively iterating over each element in the XML tree, each element was processed depending on its data. If the element was a map, then its children would be parsed as key-value pairs. If the element was a list, then its children would be added to a list. Any other element was parsed with a name and a value. All elements were added to a dictionary. Each element is guaranteed to have a name due to the serialisation assigning names to each element for each field.

For MongoDB, the PyMongo library was used to interface with the database connection [18]. Following the documentation, the tutorial was simple enough to follow to setup a connection to a local MongoDB instance. The `SubmissionCollection` class was written to provide useful database methods. Primarily, these methods are used to find, and insert data into the database easily. Example MongoDB Student Document in Code Examples shows the document data structure for a typical user.

4.2.6 Using Flask to Deliver Web Content

The back-end server must provide a service to staff and students. Enter, Flask. Flask provides a micro web framework. Flask uses function decorators to specify URL routes. The return value of the function is then displayed when requested. Flask uses a templating language known as Jinja2. Jinja2 allows HTML templates to be dynamically rendered for each request. Pairing the templates with Bootstrap and JQuery allows to present beautiful dynamic web pages for end users. The front-end web application doesn't do much currently. The end users must be able to login using their Aberystwyth University credentials. To combine LDAP server authentication with Flask seamlessly, we need another library; Flask-Login.

Flask-Login provides user session management for Flask. Using the Bootstrap sign-in template [19] and Flask-Login, the authentication system worked perfectly. A single route was used for the index. The index renders the sign-in template when using the GET request. However, when a POST request is sent, authentication is performed using the posted form values (username and password). The username and password are used for binding a new LDAP connection. Upon success, the user details are retrieved from the LDAP server and a new user model is created using the User class. The User class follows the model from Flask-Login providing the required methods. An additional method has been added to test if the user is staff. The user details are stored in the database. If the user already exists in the database, then an update is triggered instead. This will update and out-of-date details. The page is then redirect to the appropriate dashboard. Staff and students have separate dashboards. If authentication is unsuccessful then a flash message is displayed to the user and the sign-in page is displayed.

4.2.7 Dockerising the Server

Docker is a software technology providing containers. Docker provides base images or operating systems, such as Ubuntu which can be downloaded and run out of the box. Docker also allows images to be built in layers, meaning that developers can build on top of these base images and distribute their images (which can then be used as base images for others to download). A Dockerfile is a text file which contains commands to build an image. This allows for seamless automation of building Docker images and running Docker containers. A `Dockerfile` was created for the back-end server. This will install all of the required python packages using pip. The base image used is `python:3.6.4-slim`. This provides the basic Python infrastructure. All of the required files are added to the container upon running the container by mounting the directory to a volume in the container. Another `Dockerfile` was written for the MongoDB instance. The `docker-compose.yml` file is used by Docker Compose to run multiple-container applications. This allows deployment of the containers (back-end server, and MongoDB) with one command; `docker-compose up` to start, and `docker-compose down` to stop.

4.3 Retrospective

The velocity for this sprint is 25. This is a large increase from the first sprint. Again this sprint had bad estimates for stories. Initially, the sprint ended prematurely, so more stories were added to accommodate this. Alternatively, the sprint could've been closed and a new one started. Aside from the bad estimations, the work completed during this iteration was comprehensive.

Chapter 5

Iteration 2

5.1 Planning

This iteration goal was to allow students to post new submissions and view all previous submissions. Aberystwyth authentication for staff (and potentially students) will be a major stepping stone for this sprint. Below is the list of stories and their assigned story points. The list is displayed in completion order.

- Add form to upload submission in student dashboard: 2 points
- Display all previous submissions in student dashboard: 2 points

5.2 Implementation

5.2.1 Adding the Student Upload Form

A new route and template were created. This new page contains a form to upload a new submission. Only students can post new submissions, so this page is not accessible by staff. A title, module, and XML file are required. Uploading the XML file was a little more difficult due to the requirements. Following an online tutorial was sufficient enough to overcome this [20]. The file that is submitted must be an XML file. This is checked in the HTML form, and in the Flask route function. The submitted XML file is parsed and the submission is inserted into the student submissions array in the database. A flash message is displayed upon success or failure.

5.2.2 Displaying Previous Student Submissions

The dashboard has been separated into two individual templates. One for staff and one for students. With functionality in place to post new submissions, these can now be displayed to the user. Querying the database by filtering `uid`, the current students' submissions can be retrieved. The submissions array is passed to the template to render. The template displays the submissions in a table. The submissions are iterated, and each submission is added as a new table row. The

submission title and module are displayed in the columns. More data for each submission may be added in future.

5.3 Retrospective

This sprints' performance is extremely low in comparison to the first two. The sprint velocity is only 4. There were more stories initially for this sprint, but they were moved to the backlog and will be tackled in a future sprint. This was due to unforeseen health issues.

Chapter 6

Iteration 3

6.1 Planning

This iteration will mainly focus on adding unit tests to the back-end Python server. These stories were initially due for the previous sprint, but were not completed. Below is the list of stories and their assigned story points. The list is displayed in completion order.

- Show all submissions on staff dashboard: 1 point
- Add Python nose tests: 3 points

6.2 Implementation

6.2.1 Displaying Student Submissions for Staff

Displaying students' submissions on the staff dashboard, is similar to the already existing student dashboard. The only difference is to remove the student uid filter from the database query. Each submission needed to have some form of identification, so each submission was tagged with the user data. The staff dashboard shows the full name of the student in the submissions table.

6.2.2 Testing the Server

Nosetests is being used for the unit testing framework in Python 3. Nose extends unittest, which is the included in the Python standard library. Both unittest and nose are similar to JUnit. The mock library is also being used for the unit tests. *Mocking* allows replacing parts of the application, to create specific testing scenarios. Function decorators are used to specify which functions will be mocked and which proxy data will be used.

The first unit tests to be developed were the authentication methods. The LDAP3 authentication system was mocked to "bypass" logging in as a user. This allows the authentication methods to operate as if the LDAP server responded. Instead, the tests will mock the code to return the required data. If mock was not used, then real credentials would have to be used, and

the tests would have to be able to connect to the LDAP server on the Aberystwyth University network. The sign-in test is still incomplete, due to the database methods not being mocked. When a user signs in, their details are inserted into the database.

MockupDB is a Python 3 package. It is used to create a mock server for testing the MongoDB code. The approach that MockupDB is different from mocking methods. But the documentation and an online blog was used for reference [21]. The server code had to be refactored to account for the mocking of the database. Both the test environment and production environment need to use the same server instances (Flask application and MongoDB client). If not, then the tests would fail. When the tests are set up, the MongoDB client is replaced with the mocked instance.

The sign-in test will send a POST request to the Flask application (in a new thread). Whenever the database receives a query, the test must accept (or deny) the query, then send a response with the appropriate data. In this test case, a query is sent to retrieve the user data (the user that is attempting to sign-in). The test will acknowledge the query, and then send back None. None is sent because the user has not signed in previously. If the user had previously signed in, then the user details would be sent back instead - but that's another test case. Another request is received for inserting data into the database. This is upon successful authentication. The mocked LDAP server was used for this and simply returned True when authentication was tested. The insertion is acknowledged and the data that is to be inserted is validated. This validation will either fail or pass the test.

Once this first sign-in test was in place, the other test case scenarios would much be easier to write. Some of the other scenarios are, sign-in after previously signing in, and sign-in with incorrect credentials. Dashboard tests will be developed in a future sprint.

6.3 Retrospective

The issues with the previous sprint have continued into this iteration, unfortunately. The velocity is 4, which also follows the previous iteration. This was due to unforeseen health issues.

Chapter 7

Iteration 4

7.1 Planning

The mid-project demonstration date is approaching. Below is the list of stories and their assigned story points. The list is displayed in completion order. Bugs are not assigned story points.

- Bug: Submissions disappear upon sign-in
- Create architecture design: 1 point
- Create user sequence diagram: 2 points
- Start mid-project demonstration plan: 3 point
- Add invalid sign-in test: 2 points
- Add User class tests: 2 points

7.2 Implementation

7.2.1 Squashing More Bugs

When a user signs in, a bug occurs which removes all of their previously posted submissions. This was a major bug which required immediate fixing. The code responsible for this was the sign-in database operations. Upon sign-in, the database is updated with the users latest details. But along with that information is the submissions array. A new array was being inserted, and overwriting any previous data. This was changed to only occur during the users' first sign-in.

7.2.2 Planning for the Mid-Project Demo

For the mid-project demonstration, diagrams would be needed to aid with explaining the system (see chapter 12 for the final diagrams). The architecture diagram was developed to clearly show how the system components interact with one another. A large detailed sequence diagram was

developed to show the interactions between the users and each of the components. The sequence diagram was later split into multiple smaller sequence diagrams. An overview or plan for the demo was also needed. This started as a markdown document containing notes, but later was converted into a Google Slides presentation.

7.2.3 Adding User Tests for the Server

More unit tests were needed for the back-end server. The user model class is a core part of the authentication and session management system. These unit tests were not very complex and did not require mocking. The tests simple initialise new instances and ensure that the methods work correctly.

7.3 Retrospective

This sprint was much better in comparison to the past two sprints. The velocity has increased to 10. Although not as high as the first two, it is definitely an improvement.

Chapter 8

Iteration 5

8.1 Planning

The mid-project demonstration date is approaching. Below is the list of stories and their assigned story points. The list is displayed in completion order.

- Create post-processor module hello world: 2 points
- Add Docker container to post-processor: 1 point
- Create post-processor module hello world: 2 points
- Detect automatic code generation: 3 points

8.2 Implementation

8.2.1 Starting the Post-Processor

The post-processor will be developed using Python 3. Originally it was to be built-in to the back-end server. But, to allow for modularity, it will be developed separately. There will be some duplicate code, such as the database code but the post-processor does not need to directly interact with the server.

The post-processor was based on the server due to the server also being developed in Python 3. The Dockerfile was also reused and modified slightly. The `docker-compose.yml` was updated to add the post-processor service. Once the infrastructure was in place, the major features could be implemented.

8.2.2 Implementing the Watch Feature

The main feature that the post-processor needs is to be able to process new submissions that are inserted into the database. PyMongo provides a `watch` method. This method returns an iterator

which iterates over the changes. It acts as a event notification system. When a student posts a new submission, this is inserted into the database, and the watch iterator will be trigged.

As with most things in the programming world, this did not work at first. There were in fact, multiple problems faced when implementing this. Firstly, MongoDB had to be upgraded to 3.6 to be able to use the `watch` method. This was simple enough to change in the Dockerfile. Next, the database was required to be a replica set for it to work properly. Looking through plenty of online tutorials, there was lots of manual work, let alone only any Docker tutorials (or Docker Compose for that matter). At first it was thought that redundancy had to be added to the database (i.e. 3 nodes instead of 1). This proved difficult with the networking configuration. After the network issues were solved, another issue was encountered. The `replicaSet` was not able to vote for a master node. The two redundancy nodes were removed but the same problem was occurring. It turned out to be that the master/primary node had to be initialized to configure the replica set (only one time). It worked after initialization. But that was manual initialization of the master node. To automate this a bash script was written which would check if its not initialized, and then initialize it. The final issue was that deploying the application a second time would cause the replica state to fail. It turns out the hostname would change each time. The fix was to change the hostname in the `docker-compose.yml` file. This ensured the hostname was the same every time it was deployed.

In summary:

- Upgrade MongoDB to 3.6 to watch collections [22]
- Fix networking issues after upgrading to 3.6 by using `--bind_ip 0.0.0.0`
- Initialise the `replicaSet` with `mongo --eval 'rs.initiate()';`

8.2.3 Attempting to Detect Automatic Code Generation

The last story in the sprint was not completed. Implementing automatic code generation detection was thoroughly investigated. The `ActionManager` class provides a listener which is used to identify the menu action that was performed. This includes the code generation actions. The difficulty comes with identify the exact editor changes that were executed from that action. The start of the changes is known, but when is the auto generated code finished? Another option is to get the previous/last action performed when detecting editor changes. This has a similar problem, when does the action end? At this point, it is appropriate to ask how much time should be spent trying to find a solution. Should development continue without discrete identifications of source code? On the other hand, while trying to find a solution for automatic code generation, a potential solution for detecting refactoring was found. If renaming files can be detected then that would be enough for the plugin. The primary focus should then shift toward the post-processor.

8.3 Retrospective

The velocity for this sprint is 8. Most of the time during this sprint was spent on getting the `watch` method to work with MongoDB. Despite not being able to implement automatic code generation detection, it could be added in the future.

Chapter 9

Iteration 6

9.1 Planning

This iteration consists of starting this report, researching machine learning algorithms to detect plagiarism, and encrypting the XML file. Below is the list of stories and their assigned story points. The list is displayed in completion order.

- Begin initial draft of project report: 3 points
- Research machine learning techniques for plagiarism: 5 points
- Encrypt stored data: 2 points

9.2 Implementation

9.2.1 Researching Machine Learning Techniques

This iteration had less of a focus on development. Instead, this report was started, and more research for machine learning algorithms to detection plagiarism. Due to the time restriction on this project, machine learning techniques were not implemented. However, it was investigated thoroughly to find potential algorithms.

Bandara and Wijayrathna state that most plagiarism detection algorithms are based on structured methods [23]. Structured methods are based on the program structure of source code. Another method for plagiarism detection algorithms is attribute counting. This involves extracting various metrics from the source code such as line lengths, and comment frequency. These metrics are used as inputs for the machine learning classifiers. An attribute counting method for detecting plagiarism would work very well with this project. Each change could be used as the metrics. More source code identifications such as automatic code generation, and refactoring, would greatly improve the accuracy. However, the raw data of the changes would be not a viable solution. They would need to be transformed into a more suitable metric for comparison. In the future, it would be possible to implement these machine learning algorithms in the post-processor.

9.2.2 Encrypting the XML File

Previously, it was discussed that a student may attempt to generate a fake XML file. But it was recently discovered that the XML file could just as easily be modified before submission. For example, using `sed -i 's/source=CLIPBOARD/source=OTHER/' plagiarism_detection.xml` to replace all clipboard sources with other. This would cause problems with the post-processor identification. The most obvious solution is to encrypt the XML file, or at least to obfuscate the XML data. Previously it was thought that encryption was unnecessary. Due to the `PersistentStateComponent` class handling the XML serialisation and deserialisation internally, it was impossible to implement encryption at this level (i.e. just encrypting the whole XML file).

A selection of methods were attempted to effectively obfuscate the XML data. The first method used was to encrypt the file paths (the map keys in the XML file). This was successful and one way to expand on this would be to manually obfuscate each value in the map. To encrypt the map values, they would first need to be represented as a String. But not only would they need to encoded as a String, but also decoded back into an Object. Delving into the IntelliJ Plugin SDK, the `XmlSerializer` class handles serialising objects into an XML structure. Utilising this class, the map was serialised into an XML string. Now each value can be encrypted by performing 128-bit AES encryption on each XML string, then applying base64 encoding. Base64 encoding is used so that the value can be stored as a value string. This string can then be saved as usual and will be used for the persistent state. Upon loading the state, the process is simply reversed. This prevents students from modifying the XML data.

9.3 Retrospective

The velocity for this iteration is 10. The investigation of machine learning techniques has proved useful and these could be implemented in the future if possible. However, due to the time limitation, only basic plagiarism detection will be implemented. The XML file encryption has solved many issues that were present. The tracking of students file changes is now secured and this data cannot be compromised. It also provides tamper prevention, so that the data will not be altered before submission.

Chapter 10

Iteration 7

10.1 Planning

This iteration will focus on finishing the final touches to the plugin and implementing basic detection methods in the post-processor. Below is the list of stories and their assigned story points. The list is displayed in completion order.

- Detect file name changes and deletions: 2 points
- Gather sample XML files: 3 points
- Reconstruct files from change list: 2 points
- Implement copy-paste detection in post-processor: 3 points
- Implement external file change detection in post-processor: 3 points
- Create post-processed data structure: 2 points
- Update database with processed submission data: 2 points

10.2 Implementation

10.2.1 Detecting Rename Refactoring

In the previous iteration, it was mentioned that while investigating detecting automatic code generation, a potential solution was found for detecting refactoring changes. The `RefactoringEventListener` class provides an interface to listen to refactoring events. Specifically, `refactoringStarted`, `refactoringDone`, `conflictsDetected`, `undoRefactoring`. A wrapper class for the refactoring data was developed, `RefactoringData`. This class will store the before and after refactoring data. Two methods are using for applying and undoing the refactoring data. Applying or undoing the refactoring in this class will apply the changes to the appropriate `FileTracker` for the file that was modified. Currently, only renaming classes or files is supported. It was originally planned to also detect file deletions but instead it was decided to keep the file changes (the same as externally deleting a file).

10.2.2 Rebuilding Files From the Tracked Changes

Four XML sample files were collected by writing a simple Java application (a bingo caller). Two of the samples did not correctly track the beginning of the project. The `public class ... { }` declaration was not recorded. Using these sample XML files a small algorithm was developed to reconstruct each of the files using the list of changes. Listing 10.1 shows this Python algorithm below. This algorithm worked perfectly, aside from missing the class declaration. The fix for that was rather simple. It works by first checking if the file is not already being tracked. Then it will get the contents (before the change currently being made) and if the contents are not empty it will add the change as external.

```
document = ''

# Add each change to the document
for c in self.changes:
    # Get change data
    old_str = c['oldString']
    new_str = c['newString']
    offset = int(c['offset'])

    # Get the start and end of the document which shouldn't be
    ↪ modified
    start = document[:offset] if len(document) > 0 else ''
    end = document[offset + len(old_str):] if len(document) > 0
    ↪ else ''

    # Insert the new value into the document
    document = start + new_str + end

return document
```

Listing 10.1: Python code to build a string document from a list of changes

10.2.3 The Post-Processed Result Data Structure

Example MongoDB Student Document in Code Examples shows the MongoDB document for a student. The `result` value is the post-processed data. This data structure contains a set of metrics. Each tracked file has its own set of metrics. In the next sprint, these metrics will be used to calculate a plagiarism value on a linear scale.

- **diff_ratio**: This is the ratio difference between the cached file contents and the reconstructed file contents from the list of changes. This acts as an accuracy modifier.
- **frequency_total**: The total character count in the file
- **frequency_clipboard**: The character count from clipboard changes

- **frequency_external**: The character count from external changes
- **frequency_other**: The character count from any other changes (keyboard, auto code generation, etc.)
- **frequency_time_source_data**: The frequency vs. time data including source. This data will be used to plot a scatter/line graph

Matplotlib can be used to easily create graphs. Using the `frequency_time_source_data` values, a line/scatter graph of Code Frequency vs. Time can be created. One graph is created per submission. Merging the metrics for each file in a submission will allow creating a single graph for each submission. This graph will show the character additions/deletions over the time of the project. This allows easy visualisation for larger chunks of code being added (i.e. large copy/paste). Other metrics which are being generated are; total frequency (character count), clipboard frequency, external frequency, and diff ratios. The frequencies are simply the character for a source. The diff ratios is comparing the file cache to the reconstructed document (1.0 is perfect match). Currently these metrics are not being inserted back into the database and therefore not being shown on the dashboard. Updating the submission with the result data was a simple task. The user id and submission id are both used to update the submission with the new data.

10.3 Retrospective

The velocity for this sprint was 17. An increase from the past couple of iterations. The stories completed in this iteration developed the fundamentals for the post-processor detection methods. The next step will be to complete these detection methods and display the metrics in the staff dashboard. If the next iteration is as productive as this, then the post-processor and front-end web application should be completed.

Chapter 11

Iteration 8

11.1 Planning

This main goal for this iteration is to complete the post-processor and front-end web application. Finishing the detection methods implementations and displaying the metrics and chart in the staff dashboard. Below is the list of stories and their assigned story points. The list is displayed in completion order.

- Add post-processor tests: 3 points
- Add full documentation to each module: 2 points
- Add installation instructions: 2 points
- Add submission graph visuals for staff dashboard: 3 points
- Implement a plagiarism / UAC scale: 1 point

11.2 Implementation

11.2.1 Testing the Post-Processor

Due to the post-processor data structure not being well developed before implementation, unit tests were not appropriate at the time. Now that the data structure is mostly complete, it is now reasonable to add these unit tests. Currently, due to time restrictions, only one unit test was implemented. A simple input-process-output test. This test uses sample encrypted XML data, runs it through the post-processor, and a result is returned. The result is compared to the expected data.

11.2.2 Documenting the Code

Over the course of the last few sprints, code documentation has become scarce. A story was put into this sprint to accommodate for the lack of documentation, including installation instructions.

Each module was fully documented with appropriate Javadocs, docstrings, and line comments where necessary. The installation instructions were added to the `README.md` file. These instructions also included system requirements. Separate instructions are documentation for each module.

11.2.3 Plotting Charts using Pygal

Currently, Matplotlib is used for displaying the frequency vs. time chart. This only works locally, but is useful for testing the chart data. Pygal will be used to display charts in the Flask application [24]. Matplotlib and Pygal share a very similar method of building charts. This made a smooth transition to Pygal to display the FTS (Frequency Time Source) data. Matplotlib is still used for local testing of charts.

The Pygal submission charts were added to the staff dashboard. Initially they were embedded directly into the table. This took up a lot of space and was aesthetically pleasing. It was decided to add a new page which would display all submission details. This would keep the submissions table simple, providing a link to each submission detail page. The submission detail page uses this url format, `/dashboard/submission/<user_uid>/<submission_id>`. The staff dashboard table now has a link to access the submission details view. The submission details displays the metrics, the FTS chart, and a list of *large changes*. The metrics are displayed in a description list. The FTS chart is displayed as an SVG+XML image. This allows the Pygal chart to provide interactivity. This includes selecting specific sources and tool tips showing discrete values. A *large change* is a change which meets a specific size requirement. The default size considered for a change to be *large* is 200 characters. This can be configured by specifying a GET parameter in the URL. For example, `/dashboard/submission/abc12/5aca40cca326f6001046e1b8?large_change_size=91`. This url will show a specific submission where large changes are classed as having a size of 91 or greater. The total time metric for submissions was added which lead to the creation of the CPM (characters per minute) metric. The CPM is calculated by simply dividing the total frequency by the total time. The average CPM is between 190 and 200 [25]. For professional typists the average CPM is between 325 and 375. However, the CPM when using IntelliJ could be drastically higher. Due to tracking every single change, auto-completion, automatic code generation, and refactoring all count towards the CPM. This will have to be taken into account when evaluating a submission.

11.2.4 The Plagiarism Value Metric

The final metric to be added was the p_value metric (i.e. the plagiarism value). This is calculated using all the metrics available. It also has an associated colour (low value is green, high value is red). The p_value operates on a linear scale. Currently this is 0 to 40. This scale is only a guideline and may change in the future with more testing. A large range of samples would be required for a more accurate scale. The p_value equation is shown below.

$$p\_value = \frac{t}{100} \cdot d \cdot ((c+1) + (e+1)^2)$$

where t = Total time,
d = Diff ratio,
c = Clipboard change frequency,
e = External file change frequency

The clipboard and external change frequencies are added together. The external frequency is exponential because externally changing a file allows any kind of changes. The diff ratio acts as an accuracy modifier. Usually the diff ratio will be 1.0 so it will not affect the final value. The diff ratio will be less than 1.0 when one or more changes are not tracked. The p_value also has a colour associated with it. Depending on the value, the colour will gradually change. Listing 11.1 shows the algorithm below for determining the colour for a p_value. The `__P_VALUE_LIMIT` is defined as 40. But this may change in the future.

```
mod = 255 / __P_VALUE_LIMIT
r = min(255, round(p_value * mod))
g = max(0, 255 - round(p_value * mod))
b = 0
```

Listing 11.1: Python algorithm to calculate colour based on the p_value

11.3 Retrospective

This velocity for this final iteration is 11. All of the development was completed during this iteration. The post-processor and front-end web application are finished now. The following chapters in this report will include the design architecture, testing, evaluation, and conclusions.

Chapter 12

Design Architecture

As discussed in Iteration 1 in chapter 4, the non-continuous server design was chosen. The continuous server would be more complex to design and implement than the non-continuous version. This is because a constant connection between the plugin and server was required. If the network connection was interrupted or disconnected then data would not be sent to the server. This means that a fail-safe protocol would have to be implemented, such as writing the unsent data to file. This fail-safe protocol implementation would be required for the non-continuous server anyway.

The server-side of the continuous design would also need to be more sophisticated. The server would have to know when a new project is being started, which project data is being received, and when a project is completed. The non-continuous server only required a method to submit the tracked data file.

The following pages show diagrams to aid with discussing the final design of the system.

12.1 Architecture Diagram

Figure 12.1 shows the interaction between each of the modules. The plugin only needs to track file changes in the editor. This data is saved to an XML file, which the user can submit to the server using the front-end web application. This requires authentication using Aberystwyth credentials. Once the file has been submitted, the back-end server converts the XML data to JSON and stores it in the MongoDB database. The post-processor is continually monitoring the database for new student submissions to process and update the database with the result.

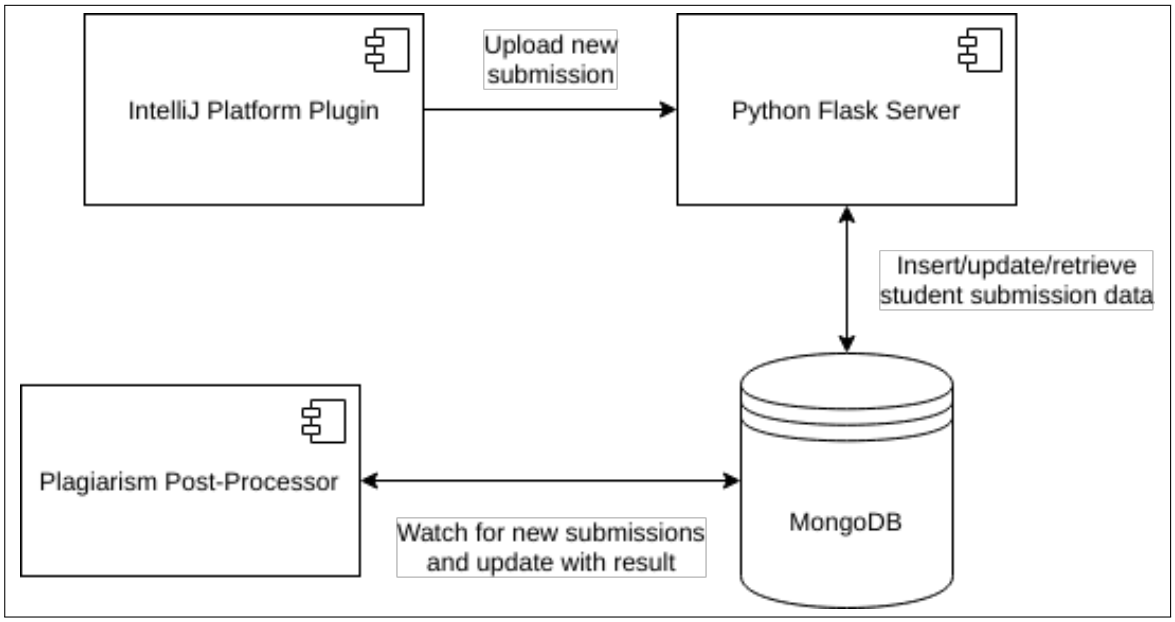

Figure 12.1: Architecture diagram

12.2 Student Plugin Interaction Sequence Diagram

Figure 12.2 shows a typical students' interaction with the plugin. Initially, when a student opens a project, the tracked files are checked for external changes. This is done by comparing each files last known contents (which is stored as a base 64 encoded value) with its current file contents. The difference is added to the list of changes for each file. A DocumentListener is then added to the project for listening to file changes. Each file change is identified and then added to the list of changes for that file. After every new change that is added, the files last known contents is updated (for checking external changes). The recorded data is saved any time a file is saved. File change tracking stops when the project is closed (or if the plugin is disabled or uninstalled).

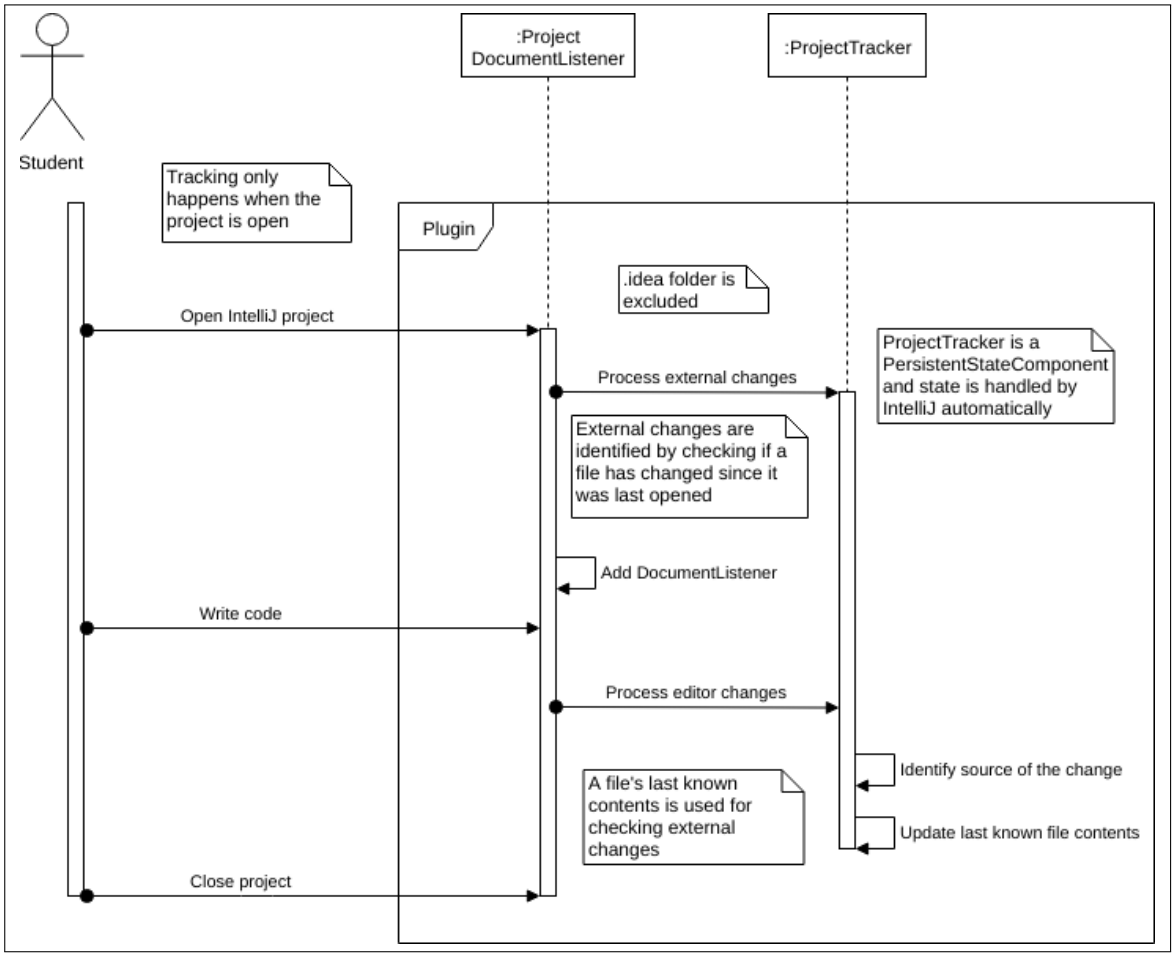

Figure 12.2: Sequence diagram showing a student interacting with the IntelliJ plugin

12.3 Student Server Interaction Sequence Diagram

Figure 12.3 shows a students' interaction with the server. Once the project is completed. The student may login to the web application using their Aberystwyth University credentials. This authorisation is achieved using the LDAP server provided by Aberystwyth University. This requires the back-end server to be running on the Aberystwyth University network. The LDAP server returns if authentication was successful or not. If successful, then the student will be redirected to their dashboard. A query is sent to the database to retrieve all of the students' submissions. These submissions are then displayed in a table in the dashboard. Students can post new project submissions by uploading the XML file with additional meta data for identification in a form. The form is submitted via POST. The XML file is decrypted and added to the students submissions in the database. The user will be notified of success or failure via a flash message.

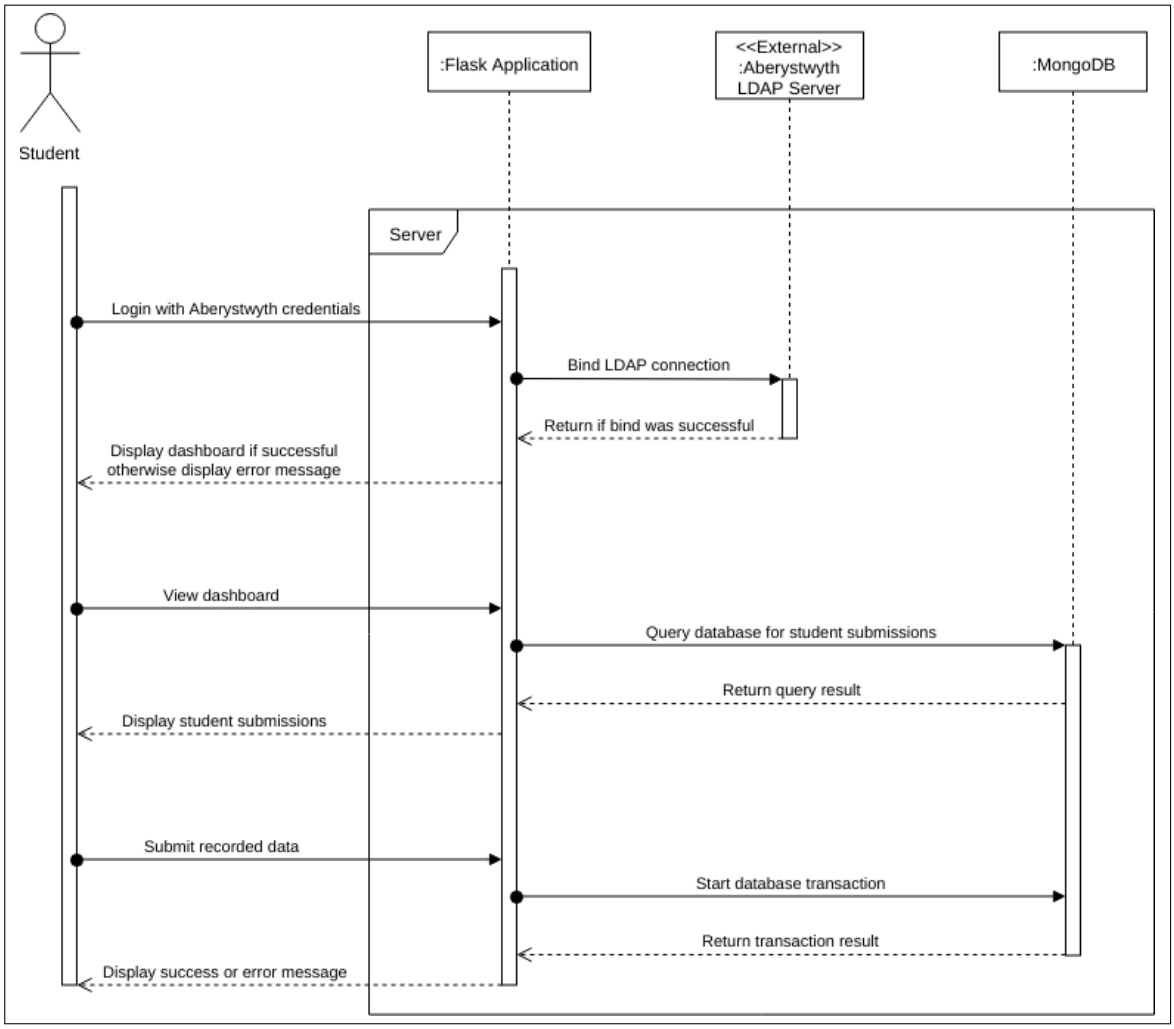

Figure 12.3: Sequence diagram showing a student interacting with the server

12.4 Post-Processor Sequence Diagram

Figure 12.4 When a student posts a new submission via the web application, the post-processor is notified. This is implemented using the `watch` method from PyMongo. The `watch` method notifies the post-processor of all changes in the database in real time. These changes must first be filtered to only identify new submissions. All new submissions are processed and then the resulting data is inserted back into the database using the original submission id and user id.

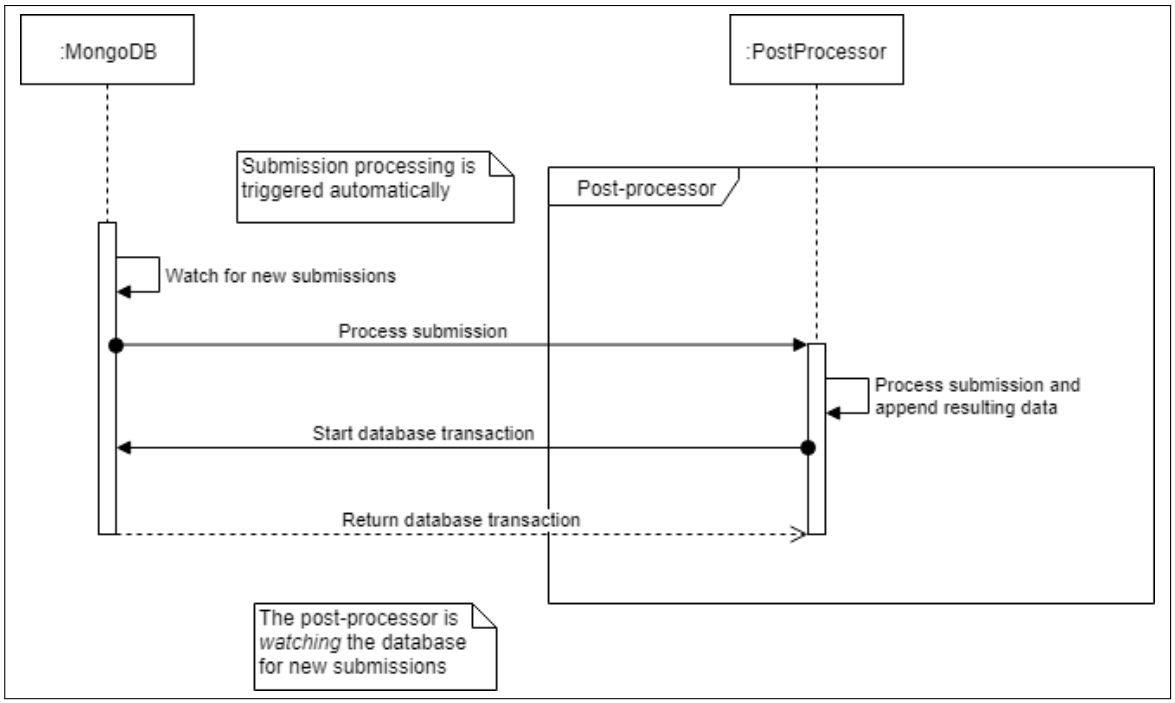

Figure 12.4: Sequence diagram for the post-processor

12.5 Data Structure Class Diagram

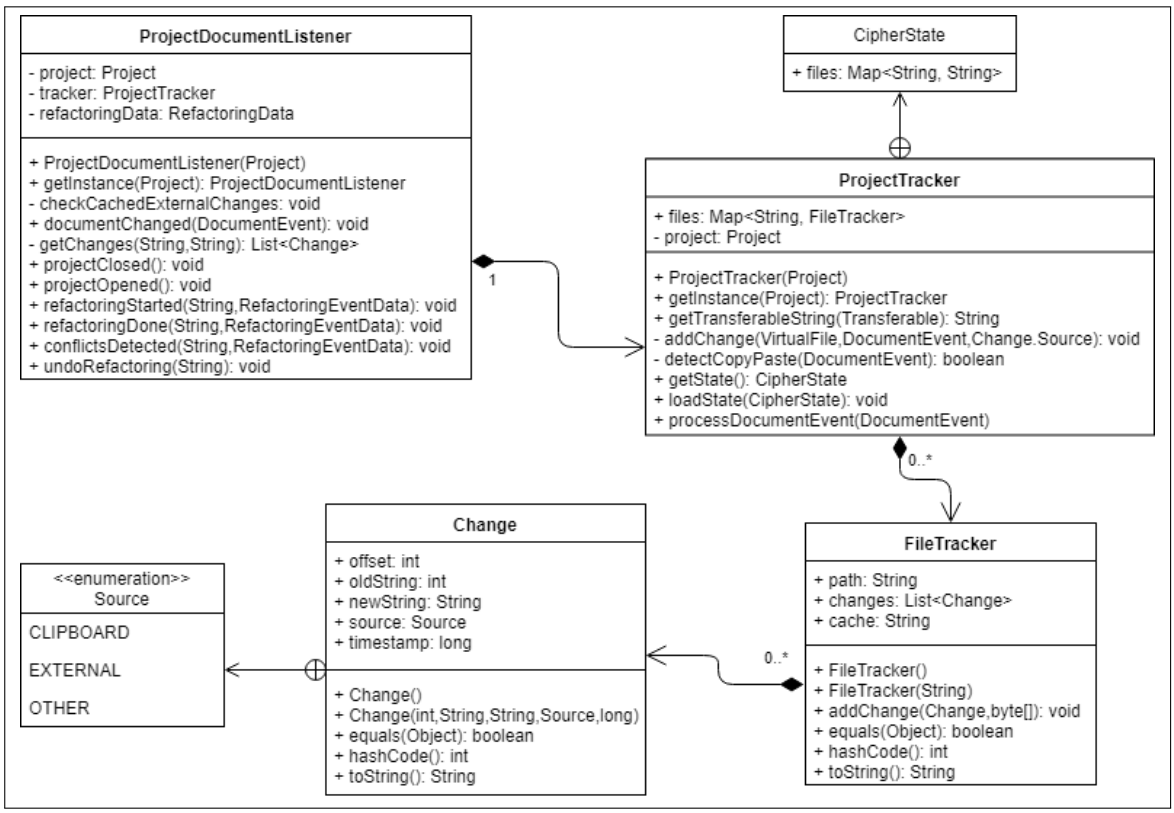

Figure 12.5: Data structure to store tracked data

12.6 Front-end Web Application

Chapter 13

Testing

13.1 Strategy

13.2 Plugin

JUnit was used for testing the plugin. The IntelliJ Plugin SDK provides a testing infrastructure. This allows testing the plugin in a headless environment. What this means is that the IDE is run without the UI to test all aspects of the plugin. Everything is loaded as usual except the UI. The SDK provides classes and methods to *emulate* user actions such as typing, pasting, clicking menu items, and clicking tool bar buttons. Emulating such actions allows testing of detecting editor changes.

Four classes are included in the tests directory. `BaseClass` provides useful assertion methods. These will check file changes for specific data. `CipherTest` is a simple test case for the 128-bit AES encryption on the tracked data. Encrypted and unencrypted sample data is used to test the both the encryption and decryption methods. `CopyPasteDetectionTest.java` is a core source detection test for detecting copy-paste in the editor. This ensures that all copy-paste actions in the editor are tracked properly. `ExternalDetectionTest` is another core source detection test for detecting external file changes. This test is a unique test because it needs to simulate externally changing a file without using the IDE editor. This works by notifying the `ProjectDocumentListener` that the project has closed, making the changes (and therefore making ”external” changes), notify the `ProjectDocumentListener` that the project is opened, which will detect the ”external” changes correctly.

13.3 Server

Nose was used for testing the back-end server. Nose extends unittest to provide extra functionality. Unittest is built-in to Python 3 and works in a similar manner to JUnit. Mock is a library used to replace parts of the system to change functionality. MockupDB is a library used to mock a MongoDB client.

`base.py` provides generic `setUp` and `tearDown` methods, as well as a method to mock or patch the Aberystwyth LDAP connection. This is useful so that the Aberystwyth LDAP server is

no directly contacted but instead is replaced with specific values to return. This removes the need to be connected to the Aberystwyth network when running the unit tests, and any username/password combination may be used for tests. `test_data.py` has all necessary testing data for the unit tests.

`test_auth.py` contains functions which test the LDAP authentication system. The LDAP connection is patched to provide the necessary user information. The scenarios that are tested are: existing user sign-in, first time user sign-in, invalid user credentials sign-in, checking if user is a staff, and checking if user is a student.

`test_dashboard.py` provides functions to test the staff and student dashboard routes. This involves sending various POST and GET requests to sign-in, and view the dashboard. The various database requests are received and the necessary data is sent as a reply to each request. The dashboard request is checked to ensure adequate data in the submissions table.

13.4 Post-Processor

Nose was also used for testing the post-processor module due to it also being developed with Python 3. MockupDB is also used for mocking the MongoDB client if necessary. The `base.py` class is very similar but does not have the function for patching the LDAP connection as this is not used in the post-processor. `test_process.py` will test the processing of a submission to ensure the result is as expected. The input as an encrypted XML sample string and the expected output is a dictionary containing the results. The input is processed and the return value is tested.

Chapter 14

Evaluation

The major requirements for this project were an IntelliJ Platform Plugin, back-end server, post-processor, front-end application, and a database. Each of these components were fully-developed. The settings GUI for the plugin was not implemented due to the using the continuous server architecture. Code generation, and code auto-completion of the source code detection types were not implemented due to the complexity of the identification process.

The primary design decision between using a continuous or non-continuous server architecture was a huge decision during this project. It impacted the method in which students would interact with the plugin and submit their tracked data. This in turn affected the back-end server internals. This decision was required before fully developing the server. If the continuous design was used then it would have most definitely increased the workload. Testing the direct connection between the plugin and server would be significantly more difficult than being able to individually test each component.

Using MongoDB was a very clear and strong design decision. If a RDBMS was to be used then the tracked data would need to be completely different from its current form. The use of Python has a clear advantage for using MongoDB. The simplicity of using dictionaries lead to simple interactions with the database.

If the project was to be written from scratch again, it would be interesting to have a greater focus on the post-processor. Concentrating more on the identification of possible cases of plagiarism would provide a much better result. It would be better due to the simplicity of the current identification algorithm being used. Integrating complex algorithms in the post-processor would allow for a more concrete result for each submission.

IntelliJ IDEA offers many ways to interact with the editor. Many were explored during the project including auto-completion, automatic code generation, and refactoring. Other methods that were not explored were using IdeaVim, and applying Git operations. IdeaVim emulates Vim in the editor. How would the use of this affect the detection methods? Would yank and paste still register as copy and paste? Using Git to stash changes, change branches, or even simply pulling from upstream can all modify files. Would these changes be tracked? If so, would this result in conflicts in the XML file? The XML file would need to be staged in each branch and commit before changing branch. Both IdeaVim, and Git could have been investigated.

Figure 14.1 shows the velocity for each sprint. The average sprint velocity is 11. Figure 14.2 shows the burndown for all iterations.

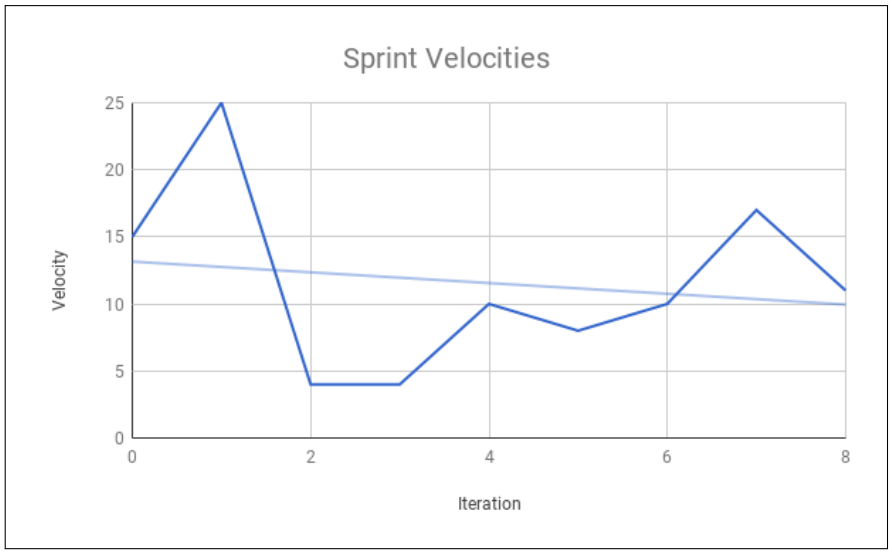

Figure 14.1: Velocity for each iteration

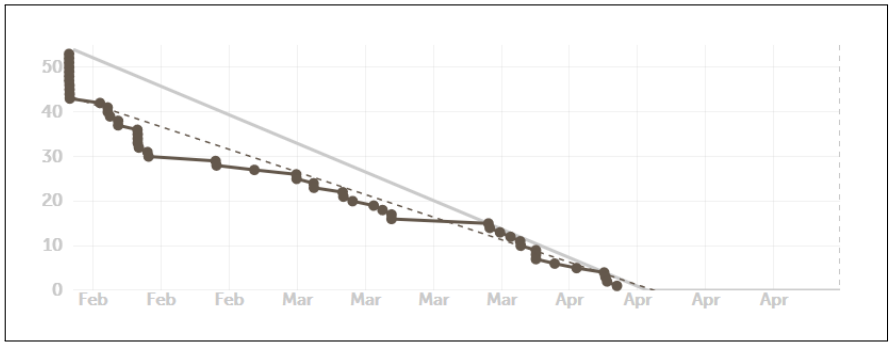

Figure 14.2: Burndown for all iterations

Chapter 15

Conclusions

15.1 TODO

1. Add screen shots - front-end web, terminal, IDE (how file changes correspond to XML changes), unit tests
2. Create web application diagrams
3. Include more code examples (Python/Java) and test examples
4. Mention open source libraries (and licenses)
5. Complete Design Architecture
6. Fix chapter/section tenses
7. Complete Acknowledgements
8. Complete Conclusions
9. Proof read

Appendices

Appendix A

Third-Party Code and Libraries

Appendix B

Ethics Submission

The ethics submission is shown on the following pages. The Ethics Application Number is: 9702.

AU Status
Undergraduate or PG Taught

Your aber.ac.uk email address
daw48@aber.ac.uk

Full Name
Darren White

Please enter the name of the person responsible for reviewing your assessment.
Chris Loftus

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment
cwl@aber.ac.uk

Supervisor or Institute Director of Research Department
cs

Module code (Only enter if you have been asked to do so)
CS39440

Proposed Study Title
MMP: IntelliJ Plugin to Aid With Plagiarism Detection. Tracks file changes in the editor of IntelliJ IDEA to help identify UAC or plagiarism. Not used with students during the course of the project. Could be used with students post-completion.

Proposed Start Date
01/02/2018

Proposed Completion Date
04/05/2018

Are you conducting a quantitative or qualitative research project?
Mixed Methods

Does your research require external ethical approval under the Health Research Authority?
No

Does your research involve animals?
No

Are you completing this form for your own research?
Yes

Does your research involve human participants?
No

Institute
IMPACS

Please provide a brief summary of your project (150 word max)
IntelliJ IDEA plugin to track file changes in the editor to help with detecting plagiarism or UAC. During the research, student participants were not involved. However, post-completion the plugin could be used in academia involving students.

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?

Not applicable

Will appropriate measures be put in place for the secure and confidential storage of data?
Yes

Does the research pose more than minimal and predictable risk to the researcher?
Not applicable

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?
No

Please include any further relevant information for this section here:

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.
Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.
Yes

Please include any further relevant information for this section here:

Appendix C

Code Examples

3.1 Example MongoDB Student Document

Below is an example MongoDB student document. The document contains the user information retrieved from the Aberystwyth University LDAP server. The submissions array contains all of the students tracked project data. Each submission entry has a one-to-one relation with the XML file. The submission `_id` is unique for each submission. The `_id` also contains the date which is useful to know when submission was posted. The submission contains input from the student, such as the title and module. The XML file that is submitted is processed and transformed into the correct formatting (JSON) which contains all of the tracked files. Each file has its path, changes, and cache. After post-processing, the result is added to the submission.

```
{
  "_id" : ObjectId("5aca40bda326f6001046e1b7"),
  "uid" : "abc12",
  "full_name" : "John Smith",
  "user_type" : "ABUG",
  "submissions" : [
    {
      "_id": ObjectId("5accc6ba7041650010aa3a37"),
      "title": "A Submission",
      "module": "CS12345",
      "files": {
        "src/Test__java": {
          "path": "src/Test.java",
          "changes": [
            {
              "offset": "0",
              "oldString": "",
              "newString": "package PACKAGE_NAME;\n\npublic
              ↪ class Test {\n}\n",
              "source": "OTHER",
              "timestamp": "1522599829159"
```

```
                }
            ],
            "cache": "cHVibGljIGNsYXNzIFRlc3QgewoKfQ=="
          }
        },
        "result": {
          "src/Test__java": {
            "diff_ratio": 1,
            "frequency_total": 45,
            "frequency_clipboard": 0,
            "frequency_external": 0,
            "frequency_other": 45,
            "frequency_time_source_data": [
              {
                "f": 45,
                "s": "OTHER",
                "t": NumberLong("1522599829159")
              }
            ]
          }
        }
      }
    ]
}
```

Listing C.1: Example student JSON MongoDB document

Annotated Bibliography

[1] Plagiarism.org. (2018) Plagiarism: Facts & stats - plagiarism.org. Last accessed: 2018-04-06. [Online]. Available: http://plagiarism.org/article/plagiarism-facts-and-stats

This web page contains results from surveys performed in schools on plagiarism. The outcomes from these surveys show how many students commit plagiarism.

[2] A. University. (2018) Aberystwyth university - regulation on unacceptable academic practice. Last accessed: 2018-04-06. [Online]. Available: https://www.aber.ac.uk/en/aqro/handbook/regulations/uap/

This shows Aberystwyth University's regulation on UAP and plagiarism. It describes the definition of UAP and the multiple forms it can take.

[3] Plagiarism.org. (2018) What is plagiarism? - plagiarism.org. Last accessed: 2018-04-06. [Online]. Available: http://plagiarism.org/article/what-is-plagiarism

This web page has detailed information on what exactly plagiarism is. It states what the definition of plagiarism is and example forms of plagiarism.

[4] Turnitin. (2018) Turnitin - the plagiarism spectrum. Last accessed: 2018-04-06. [Online]. Available: http://turnitin.com/assets/en_us/media/plagiarism-spectrum/

This web page shows a list of 10 different ways to plagiarise with example texts.

[5] P. Clough and D. O. I. Studies, "Old and new challenges in automatic plagiarism detection," in *National Plagiarism Advisory Service, 2003; http://ir.shef.ac.uk/cloughie/index.html*, 2003, pp. 391–407.

In-depth details on lexical and structural changes in program code plagiarism.

[6] A. Parker and J. Hamblen, "Computer algorithms for plagiarism detection," *IEEE Transactions on Education*, vol. 32, no. 2, pp. 94–99, may 1989. [Online]. Available: https://doi.org/10.1109/13.28038

Details on the plagiarism spectrum. The different levels of plagiarism. Level 0 has no changes and level 6 has the control logic changed.

[7] JetBrains. (2018) What is the intellij platform? Last accessed: 2018-04-07. [Online]. Available: http://www.jetbrains.org/intellij/sdk/docs/intro/intellij_platform.html

This page describes what the IntelliJ Platform is.

[8] R. Lukashenko, V. Graudina, and J. Grundspenkis, "Computer-based plagiarism detection methods and tools," in *Proceedings of the 2007 international conference on Computer systems and technologies - CompSysTech '07*. ACM Press, 2007. [Online]. Available: https://doi.org/10.1145/1330598.1330642

Useful attribute table containing information on tools for detecting plagiarism such as Turnitin, and MOSS.

[9] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: local algorithms for document fingerprinting," in *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data - SIGMOD '03*. ACM Press, 2003. [Online]. Available: https://doi.org/10.1145/872757.872770

This went into detail on the ideas behind MOSS, such as document fingerprints and k-grams.

[10] L. Prechelt and G. Malpohl, "Finding plagiarisms among a set of programs with jplag," vol. 8, 03 2003.

Detailed information on how JPlag detects plagiarism using tokens.

[11] M. Wise, "Yap3: Improved detection of similarities in computer program and other texts," vol. 28, 04 1996.

In-depth detail on the RKR-GST algorithm used by YAP3 and the comparison between the YAP versions.

[12] JetBrains. (2018) Creating your first plugin. Last accessed: 2018-04-09. [Online]. Available: https://www.jetbrains.org/intellij/sdk/docs/basics/getting_started.html

The JetBrains tutorial for developing an IntelliJ Plugin.

[13] ——. (2018) Jetbrains/intellij-community: Intellij idea community edition. Last accessed: 2018-01-31. [Online]. Available: https://github.com/jetBrains/intellij-community

The IntelliJ IDEA Community Edition GitHub repository provides code for plugins. This helped with understanding the IntelliJ Platform Plugin SDK.

[14] GitHub. (2018) About milestones - user documentation. Last accessed: 2018-04-06. [Online]. Available: https://help.github.com/articles/about-milestones/

Describes using issues and milestones on GitHub.

[15] ——. (2018) Closing issues using keywords - user documentation. Last accessed: 2018-04-06. [Online]. Available: https://help.github.com/articles/closing-issues-using-keywords/

Explains how to close issues using commit messages by including specific keywords.

[16] JetBrains. (2018) Plugin components / intellij platform sdk devguide. Last accessed: 2018-04-17. [Online]. Available: https://www.jetbrains.org/intellij/sdk/docs/basics/plugin_structure/plugin_components.html

Information on project components and the differences between application, project, and module level components.

[17] G. Cannata. (2018) Welcome to ldap3's documentation - ldap3 2.5 documentation. Last accessed: 2018-04-19. [Online]. Available: http://ldap3.readthedocs.io/

The LDAP3 library documentation for Python 3.

[18] B. Hackett. (2018) Pymongo 3.6.1 documentation - pymongo 3.6.1 documentation. Last accessed: 2018-04-19. [Online]. Available: https://api.mongodb.com/python/current/

The PyMongo library documentation for Python 3 and MongoDB.

[19] Bootstrap. (2018) Signin template for bootstrap. Last accessed: 2018-04-20. [Online]. Available: https://getbootstrap.com/docs/3.3/examples/signin/

A signin template using Bootstrap. This uses a form to submit a username and password in a form.

[20] Flask. (2018) Uploading files - flask documentation (0.12). Last accessed: 2018-02-27. [Online]. Available: http://flask.pocoo.org/docs/0.12/patterns/fileuploads/

Flask documentation containing a detailed tutorial on uploading files with a HTML form.

[21] A. J. J. Davis. (2013) Test mongodb failure scenarios with mockupdb. Last accessed: 2018-03-04. [Online]. Available: https://emptysqua.re/blog/test-mongodb-failures-mockupdb/

A blog post on using MockupDB with MongoDB in Python 3. The online documentation was not straight-forward and this cleared up many questions and any confusion. Seeing this real example helped massively when developing the back-end server unit tests.

[22] MongoDB. (2018) Change streams - mongodb manual 3.6. Last accessed: 2018-04-22. [Online]. Available: https://docs.mongodb.com/manual/changeStreams/

MongoDB documentation for change streams which are required for using the watch method on a collection. These are only supported in 3.6.

[23] U. Bandara and G. Wijayrathna, "Detection of source code plagiarism using machine learning approach," *International Journal of Computer Theory and Engineering*, pp. 674–678, 2012. [Online]. Available: https://doi.org/10.7763/ijcte.2012.v4.555

This article used three machine learning techniques to predicate the other of source code files. Source code metrics such as line length, and comments frequency were used as inputs to the machine learning classifiers.

[24] M. Driscoll. (2018) Using pygal graphs in flask — the mouse vs. the python. Last accessed: 2018-04-23. [Online]. Available: https://www.blog.pythonlibrary.org/2015/04/16/using-pygal-graphs-in-flask/

A tutorial using pyGal for displaying charts in a Flask application.

[25] LiveChat. (2018) Free typing speed test - check your typing skills. Last accessed: 2018-04-24. [Online]. Available: https://www.livechatinc.com/typing-speed-test/

This website provides a typing speed test. It also includes the average WPM (words per minute and CPM (characters per minute).

[26] P. S, R. R, and S. B. B, "A survey on plagiarism detection," *International Journal of Computer Applications*, vol. 86, no. 19, pp. 21–23, jan 2014. [Online]. Available: https://doi.org/10.5120/15104-3428