

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 12, 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 12, 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 12, 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 shows the research and development of a new tool which will detect how the code was written. This tool will be an IntelliJ IDEA plugin and will track file changes in the editor. The tracked data will give more of an insight into how plagiarism evolves over the course of development. This method of detection would allow direct identification of the specific pieces of code that were plagiarised.

CONTENTS

1	Introduction	1
1.1	Overview	1
1.2	Plagiarism and Unacceptable Academic Practice	1
2	Background	3
2.1	Existing Systems	3
2.2	IntelliJ Plugin SDK	4
2.3	Back-end Server	4
2.3.1	Post-processor	5
2.4	Analysis	5
2.4.1	Continuous back-end server	5
2.4.2	Non-continuous back-end server	6
2.4.3	Requirements	6
2.5	Process	7
3	Iteration 0	9
3.1	Planning	9
3.2	Implementation	9
3.3	Retrospective	9
4	Iteration 1	10
4.1	Planning	10
4.2	Implementation	10
4.3	Retrospective	10
5	Iteration 2	11
5.1	Planning	11
5.2	Implementation	11
5.3	Retrospective	11
6	Iteration 3	12
6.1	Planning	12
6.2	Implementation	12
6.3	Retrospective	12
7	Iteration 4	13
7.1	Planning	13
7.2	Implementation	13
7.3	Retrospective	13
8	Iteration 5	14
8.1	Planning	14
8.2	Implementation	14
8.3	Retrospective	14
9	Iteration 6	15
9.1	Planning	15

9.2	Implementation	15
9.3	Retrospective	15
10	Iteration 7	16
10.1	Planning	16
10.2	Implementation	16
10.3	Retrospective	16
11	Iteration 8	17
11.1	Planning	17
11.2	Implementation	17
11.3	Retrospective	17
12	Design Architecture	18
12.1	Final Design	18
12.1.1	Data structure	19
13	Testing	21
13.1	Plugin	21
13.2	Server	21
13.3	Post-Processor	21
14	Evaluation	22
15	Conclusions	23
	Appendices	24
A	Third-Party Code and Libraries	25
B	Code Examples	26
	Annotated Bibliography	27

LIST OF FIGURES

12.1	Architecture diagram (reminder to update this figure with descriptions for each component - maybe split it up into multiple smaller diagrams)	19
12.2	Student sequence diagram (reminder to update this figure with correct labels) . .	20

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 the 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.

Chapter 2

Background

This project introduces three 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 the capability to add and develop plugins. These plugins are developed using the IntelliJ Platform [7]. 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.

“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 allow 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

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 is 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.

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 comprehensive list of the possible technologies that could be used for the back-end server.

- **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 a SQL or NoSQL database will be used to store recorded data for users.

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. This 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 discussed more in detail in Iteration 1). Both back-end server scenarios are described below. 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

- **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 lecturers (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 lecturers to use (and possibly students)
- * Authentication page for lectures (and possibly students)
- * Display student project submission data
- * Lecturers 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 I choose to use for this project was an agile methodology, scrum. Scrum uses short iterations, each consisting of planning at the beginning, implementation, review, and then retrospective at the end. Weekly meetings on Mondays with my supervisor were also 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.

Chapter 3

Iteration 0

3.1 Planning

3.2 Implementation

3.3 Retrospective

Chapter 4

Iteration 1

4.1 Planning

4.2 Implementation

4.3 Retrospective

Chapter 5

Iteration 2

5.1 Planning

5.2 Implementation

5.3 Retrospective

Chapter 6

Iteration 3

6.1 Planning

6.2 Implementation

6.3 Retrospective

Chapter 7

Iteration 4

7.1 Planning

7.2 Implementation

7.3 Retrospective

Chapter 8

Iteration 5

8.1 Planning

8.2 Implementation

8.3 Retrospective

Chapter 9

Iteration 6

9.1 Planning

9.2 Implementation

9.3 Retrospective

Chapter 10

Iteration 7

10.1 Planning

10.2 Implementation

10.3 Retrospective

Chapter 11

Iteration 8

11.1 Planning

11.2 Implementation

11.3 Retrospective

Chapter 12

Design Architecture

Should I put this before the iterations or after?

As discussed in Iteration 1, 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.

12.1 Final Design

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 12.2 shows a typical students' interaction with each component. 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).

12.1.1 Data structure

TODO: Make class diagram for the data structure

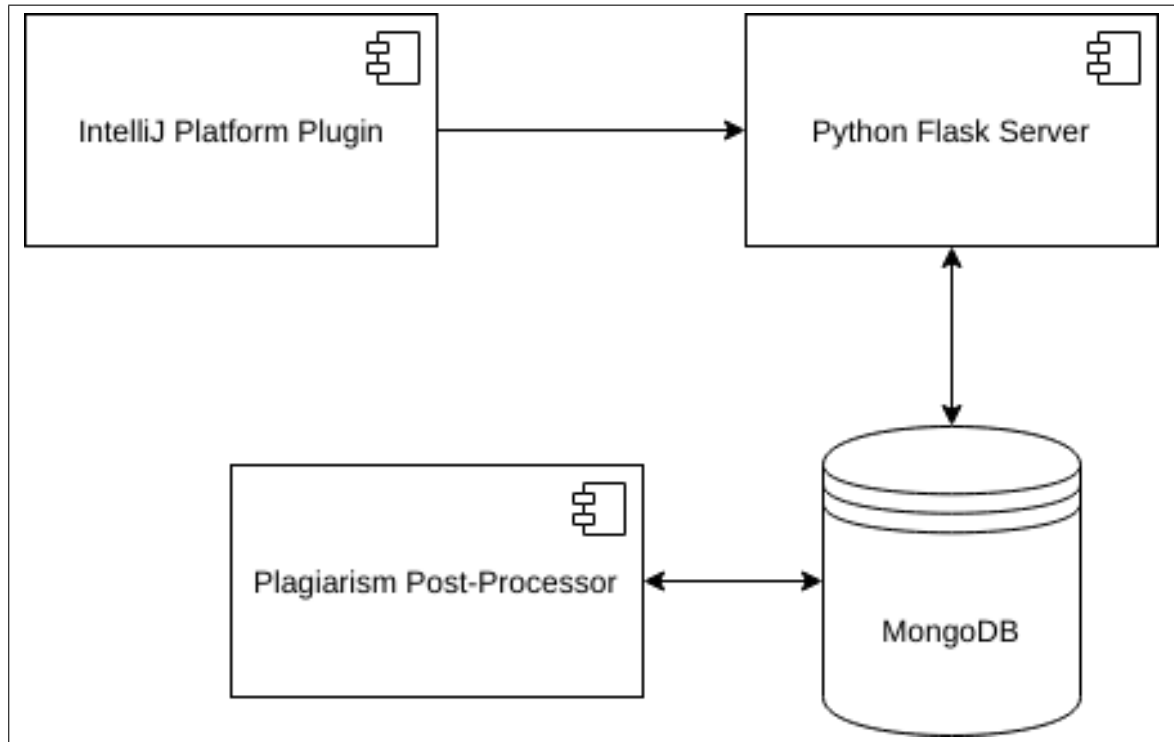


Figure 12.1: Architecture diagram (reminder to update this figure with descriptions for each component - maybe split it up into multiple smaller diagrams)

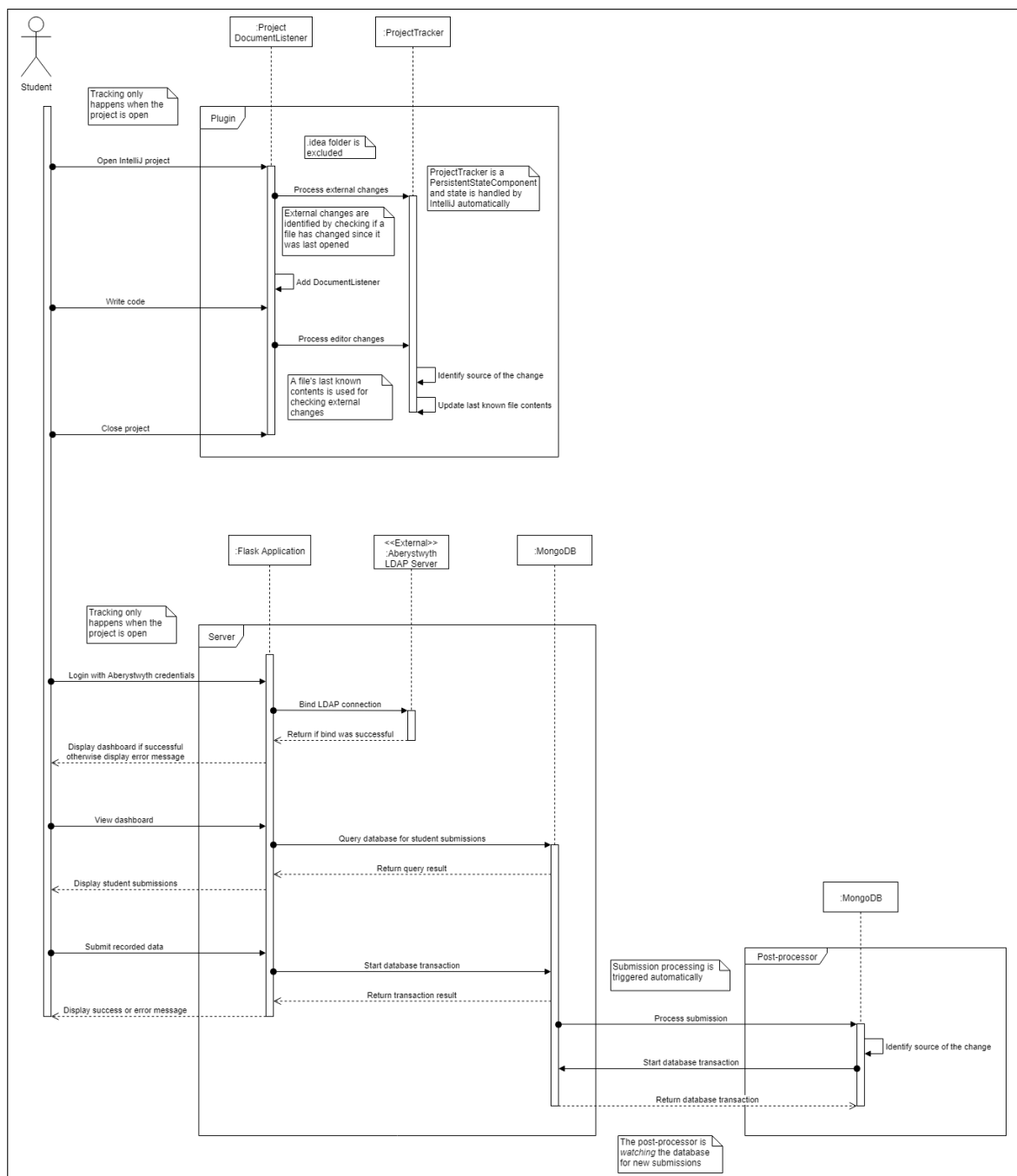


Figure 12.2: Student sequence diagram (reminder to update this figure with correct labels)

Chapter 13

Testing

13.1 Plugin

13.2 Server

13.3 Post-Processor

Chapter 14

Evaluation

Chapter 15

Conclusions

Questions:

- What tenses should we use for chapters? past, current, or future? Or does it depend on the chapter and context? - Should we mention open source licenses for third party libraries?

Appendices

Appendix A

Third-Party Code and Libraries

Appendix B

Code Examples

Annotated Bibliography

- [1] Plagiarism.org. (2018) Plagiarism: Facts & stats - plagiarism.org. [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. [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. [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. [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? [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. Gaudina, 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. [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. [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. [Online]. Available: <https://help.github.com/articles/about-milestones/>

Describes using issues and milestones on GitHub.

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

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

- [16] U. Bandara and G. Wijayathna, “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>

- [17] 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>

- [18] Flask. (2018) Uploading files - flask documentation (0.12). [Online]. Available: <http://flask.pocoo.org/docs/0.12/patterns/fileuploads/>

- [19] A. J. J. Davis. (2013) Test mongodb failure scenarios with mockupdb. [Online]. Available: <https://emptysqua.re/blog/test-mongodb-failures-mockupdb/>