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 21, 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 21, 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 21, 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

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 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 to 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 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

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 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. 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 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 to 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 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 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.

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

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.

As found during my earlier investigation of the IntelliJ Plugin SDK, the TypedHandler and TypedHandlerDelegate can be used to listen 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.

The `Change` class is the core of the data structure implementation (see Table 3.1 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 Table 3.2 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)

Table 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)

Table 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.

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`.

- 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 determining the differences between the old and new file contents.

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

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.

The options for adding units tests to the plugin were limited to one. 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(String, int)` and `assertOneChangeMatches(String, Predicate<Change>)`. 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.

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.

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.

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. section 3.1 in Code Examples shows the document data structure for a typical user.

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.

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

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.

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 circumstances.

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

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.

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.

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

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.

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.

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 much as the first two, it is definitely an improvement.

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

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 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 12.3 shows the 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 re-

turns 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 12.4 *NOTE: Not sure how much detail to do on the post-processor here.* Upon posting a new submission, the post-processor is trigged. The new submission is processed and the result is inserted back into the database.

TODO: Talk about the front-end design too. Make a diagram for the front-end application design.

12.1.1 Data structure

TODO: Make class diagram for the data structure

12.1.2 Front-end web application

TODO: Create web app diagrams

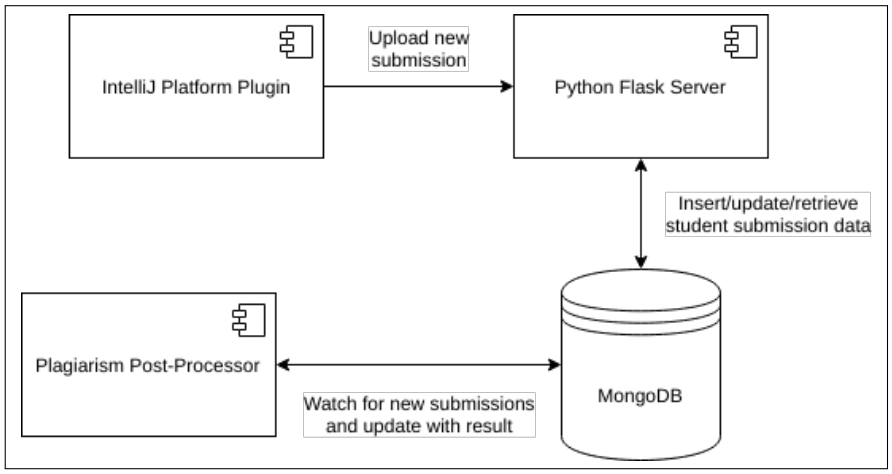

Figure 12.1: Architecture diagram (reminder to update this figure with descriptions for each component)

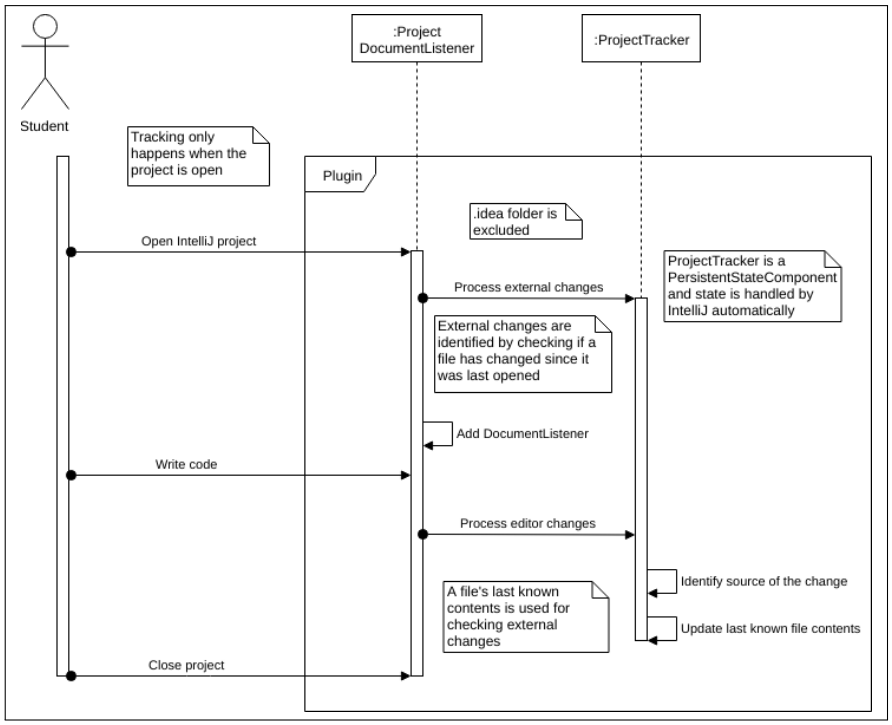

Figure 12.2: Sequence diagram showing a student interacting with the IntelliJ plugin

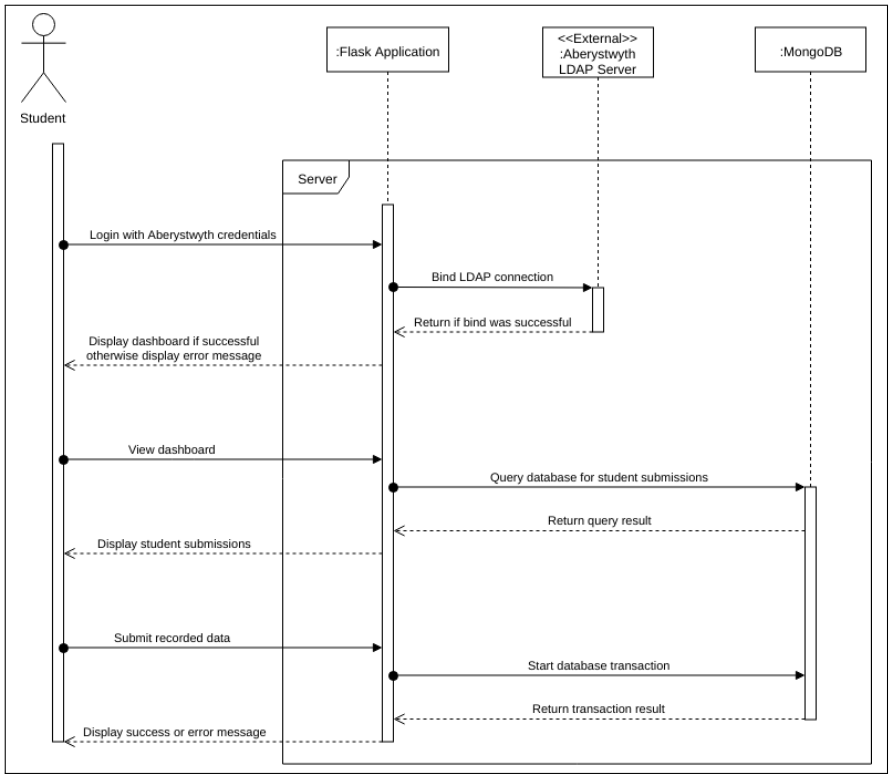

Figure 12.3: Sequence diagram showing a student interacting with the server

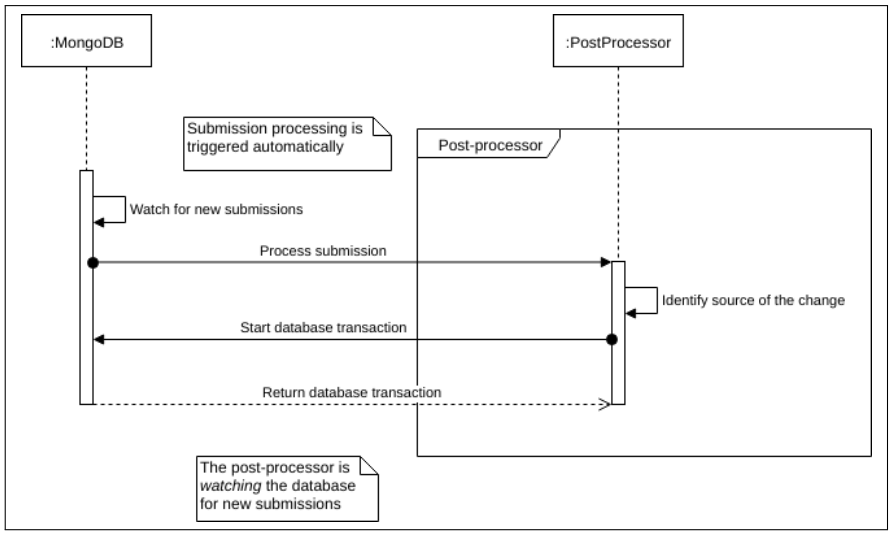

Figure 12.4: Sequence diagram for the post-processor

Chapter 13

Testing

13.1 Plugin

13.2 Server

13.3 Post-Processor

Chapter 14

Evaluation

Chapter 15

Conclusions

15.1 TODO: Notes

- Fix chapter/section tenses
- Add sprint velocity charts
- Add burndown charts

15.2 TODO: Questions

- Should I merge iterations 2 and 3? This is when I had the flu and an ear infection.
- Should we mention open source licenses for third party libraries?
- Should I split up the implementation of the iterations using subsections?
- Should I discuss bugs encountered in iterations?

Appendices

Appendix A

Third-Party Code and Libraries

Appendix B

Ethics Submission

The ethics submission is shown on the following pages.

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")
              }
            ]
          }
        }
      }
    ]
}
```

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. 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. [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] JetBrains. (2018) Plugin components / intellij platform sdk devguide. [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. [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. [Online]. Available: https://api.mongodb.com/python/current/

The PyMongo library documentation for Python 3 and MongoDB.

[19] Bootstrap. (2018) Signin template for bootstrap. [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). [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. [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] 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

[23] 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