

IFB399 – Capstone Project Assessment 2 – Final Project Report

[Abstract](#)

This document outlines the Capstone Project Phase 2 progression in terms of the project overview, the development progress made in the second phase and the artefacts that we will present and deliver. This project aims to create software that allows users to observe and analyse a program's development using software metrics. Our goals are to allow users to view metric analysis data generated from multiple versions of one program. The program analyses imported versions before displaying how data points changed over the development of a program, and provides an extensible framework for metric and data analysis.

Team Heck'n'Tech | Semester 2 2018

Contents

Introduction	2
Project Overview	2
Progress this Semester	3
The Artefact	5
Architecture	5
Technical Description	5
Functionality	8
Future development	12
References	14
Appendix 1 - User Stories	15
Appendix 2 - Project Delivery Agreement	17
Appendix 3 – UI Testing 8-10-18	23
Appendix 4 - UI Testing 15-10-18	32
Appendix 5 - UI Testing 21-10-18	42
Appendix 6 - Metric Functionality Testing	52
Code Excerpt 1	55
Code Excerpt 2	57

Introduction

Every day new software is created, and existing software continues to be updated. However, as software continues to evolve, it becomes more difficult and complex for developers to maintain code quality through the changes. This has led software developers to create ways to manage the complexity of software projects. Formerly, to keep track of changes software developers used traditional testing, which required them to manually check the program state. This method is useful at first but as software grows more complex/becomes a multi-tier infrastructure, it is no longer an effective solution. Therefore, software metrics are employed to measure software characteristics and help developers to identify issues more easily.

This project was proposed by our client Professor Colin Fidge of QUT, whose disciplines are *Computation Theory and Mathematics*, and *Computer Software*. Professor Fidge requested a program capable of analysing multiple versions of a computer program and revealing how the program became more complex during its development. He suggested this be achieved by applying different metrics to the code. In response, we created MAIJ: Metric Analysis and Identification for Java.

Last year, Professor Fidge attended a Seminar about Software evolution. This seminar inspired him to request the program. He states that he is interested in understanding how programs evolve over time and across multiple versions, and remarked that MAIJ is essentially a “framework” which will allow users to do just that. He also mentioned that this program may help track program security, for example by being used to track entry points. Additionally, it may serve some use in educational contexts by allowing teachers to highlight undesirable coding trends which should be avoided, and good practices which should be emulated. MAIJ will offer developers a new perspective on their programs, and a greater understanding of how their code evolves over time.

Project Overview

This project aims to help people who want to understand how a program evolves and is maintained over time (including what changed and why). A comparison using software metrics allows users to identify software development trends as well as the relationship between the trends and the code complexity.

The software metrics used in this project until this point are lines of code, cyclomatic complexity, number of methods, coupling, max nesting and number of statements. All these metrics are mentioned and explained in *capstone project phase 1 report*.

The program is designed with a user-friendly UI and an extendable framework allowing users to input more metrics in the future. The following is a description of the general program control flow:

Step 1 – The user selects directories of code that they wish to import into the program. This choice from the user acts as the version selection mechanism, allowing users to specify exactly what they wish each version to contain.

Step 2 – Once a user has read in the data they wish to analyse, they navigate to the metric analysis window and select the metrics that they wish to apply to the program. Once the user has selected the metrics and run the analysis, the points of interest and results table become accessible.

Step 3 – A user can use the Overview tab to view the data points for all the metrics analysed, and can view by project, package, class and method, and choose the metrics that they wish to view at each version level. The table allows the user to view the changes in data across all versions of the program.

Step 4 – A user can view the source code for all the versions imported into the program in the Project tab. Inside, users can select elements of the programs such as classes or individual methods and have those displayed. By selecting a metric at the top of the window, users can view the contribution points for each metric visually in the code, making it easy to identify patterns and changes.

Step 5 – Finally, a user may wish to export their performed analysis for further data refinement in a more powerful data analytics tool such as excel, MATLAB, or R, so they may export whichever version level of data that they wish to a CSV file and read it natively into other programs.

Error handling: Every element of the GUI was tested thoroughly through interaction testing on every UI element, to ensure that a user could not crash the program or cause unintended behaviour within the system. This will be described later in the document, with a full description and documentation in appendices [3](#), [4](#) and [5](#).

Progress this Semester

In the second phase of development, the team focused on developing the core functionality of the program. After researching, we decide to use Java for the program, to make it easier for future development since Java is a common language, object-oriented and platform independent (IBM, 2018).

The team aimed to implement the user stories that formed the core functionality of the program, meaning anything with a *MUST* level priority, plus as many additional extras as we could achieve within the time frame. These are explicitly detailed in both the Delivery Document ([Appendix 2](#)), and the User Stories Spreadsheet ([Appendix 1](#)), however a brief list of these features is listed below:

- Metric Definition and code contribution mark-up
- Version Selection
- Extensible framework
- Data exportation
- Data viewing
- Clean GUI
- Modular metric selection

After successfully implementing the metrics which we set out in phase 1 (lines of code, cyclomatic complexity, number of methods, coupling, max nesting and number of statements), Colin asked us to perform further research on other metrics and see if we could find and implement them.

At this point in development we branched out into other metrics that could be very insightful into the nature of a program. The extended metrics chosen were Fanout, Number of Accessor Methods, and Access to Foreign Data. A more complete description of these methods is described in [Appendix 6](#).

At the end of semester one we defined two key progression points for the project. First was an initial alpha release in week six which contained most of the functionality required for extremely basic use of the program. This included reading in data, performing analysis, and viewing the raw output in a very primitive user interface. The target goals were not very specific about what features should be contained within this delivery because we anticipated much would be learnt during this time. We explored the realistic project scope, using the time as a feasibility analysis and as a proof of concept. The primary intention was to have a working system which demonstrated not only the viability of the concept, but also provided insight into the difficulty of the scope we initially planned for with the client.

Once this goal was met in week 6 the scope of the project was examined, and it was found that many of the initial planned user stories and interactions were still feasible and remained in the development plan. However, some features were cut in favour of polishing other, more important features. This was discussed with the client, and we redefined and clarified exactly which features we wished to deliver. These decisions were again discussed with the client, as well as the tutor, to reach an outcome all parties were satisfied with, and finalised inside the delivery document ([Appendix 2](#)).

This settled the scope of the project to be delivered in the second release in week 13. This is the final version of the project to be completed during IFB399. This release met the planned expectations that were outlined midway through the semester after release one and focused heavily on user experience and user control. A more complete overview of the final deliverable artefact will be provided later in this document.

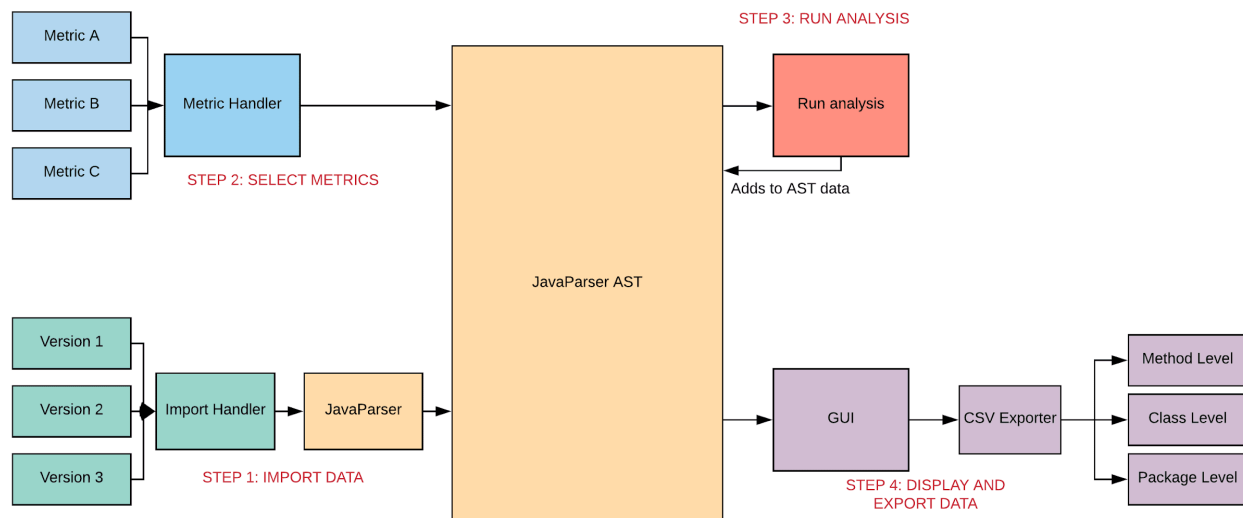
The only significant challenge presented that threatened the progress of the project was the extensive time taken updating the user interface. What we initially estimated to be a minor change to front end implementation ended up costing us a large amount of development time and resources. That detracted from time that we had initially budgeted for other project features, such as automated unit testing and a larger focus on small user interface interactions, such as button tooltips. While the core functionality of the program didn't change, and all user stories intended to be delivered were included in the final release, testing of the program was shortened. What was initially intended was a suite of automated unit tests for the metric algorithms to perform against. This ended up being performed manually. The differences will be discussed later in the document.

The Artefact

Architecture

The program largely consists of a User interface built around interacting with an Abstract Source Tree, hereafter referred to as the AST. This is tree built by the JavaParser library that we implemented to read in the data of any directory it was pointed at. The files are read into the tree and implemented as nodes in the tree. When metric analysis is performed against the tree, users can specify the node levels they wish to operate against. Once analysis has been performed, the values and data, as well as the contribution points for the metrics are stored in the nodes along with the raw data. This means that all data is centralised around the AST in the system.

To support this, the GUI largely operates around reading in this data, performing analysis, and accessing the data stored in the AST to display it to the user, or export it for use in other data analysis tools. The diagram below shows a clearer view of the data structure of the components reading in and out of the AST, the core data structure of the project.



The above diagram more visually displays the way that the program is segmented, and the way that these segments interact. While there are small helper classes and packages not explicitly mentioned, this diagram shows the core segments that form the bulk of the data operations and manipulation that is performed on data, as well as a more visual display of the way that the control flow of the program operates.

Technical Description

As discussed just above, the large component of data storage and interactions on this data formed the core of development and computational complexity. The JavaParser library (JavaParser.org, 2018), an open source library for reading Java code into a program in the form of a tree was chosen early in the project as the most viable option for reading code and treating it in a way that provides consistent and hierarchical access to data across nodes.

```

/**
 * Generates a combined heirarchy across all versions, in a way that may be used to get a particular node from the ast storage.
 */
private void generateAggregateHeirarchy() {
    aggregateHeirarchy = new TreeMap<String, SortedMap<String, List<String>>>();

    for( Double version : astStorage.keySet()) {
        for( PackageAST packAST : astStorage.get(version).getPackages().values()) {
            if(!aggregateHeirarchy.containsKey(packAST.getIdentifier())) {
                aggregateHeirarchy.put(packAST.getIdentifier(), new TreeMap<String, List<String>>());
            }

            for( ClassAST clasAST : packAST.getClasses().values()) {
                if(!aggregateHeirarchy.get(packAST.getIdentifier()).containsKey(clasAST.getIdentifier())) {
                    aggregateHeirarchy.get(packAST.getIdentifier()).put(clasAST.getIdentifier(), new ArrayList<String>());
                }

                for(MethodAST methAST : clasAST.getMethods().values()) {
                    if(!aggregateHeirarchy.get(packAST.getIdentifier()).get(clasAST.getIdentifier()).contains(methAST.getIdentifier())) {
                        aggregateHeirarchy.get(packAST.getIdentifier()).get(clasAST.getIdentifier()).add(methAST.getIdentifier());
                    }
                }
            }
        }
    }
}

```

The above code snippet from the class ASTHandler inside the ASTManager package shows the way that data is structured through the AST within our program, defining node levels, that provide easy access to different levels of code throughout the versions. This hierarchy was used to access the AST throughout the rest of the program, as this is generated on importation of files.

```

public class AnalysisHandler {
    private static final Map<String,Class<?>> registeredMetrics;
    static {
        Map<String, Class<?>> aMap = new HashMap<String, Class<?>>();
        aMap.put(new ATFDMetric().getIdentifier(), ATFDMetric.class);
        aMap.put(new CycloMetric().getIdentifier(), CycloMetric.class);
        aMap.put(new LOCMetric().getIdentifier(), LOCMetric.class);
        aMap.put(new MaxNestingMetric().getIdentifier(), MaxNestingMetric.class);
        aMap.put(new NOPAMetric().getIdentifier(), NOPAMetric.class);
        aMap.put(new NOSMetric().getIdentifier(), NOSMetric.class);
        aMap.put(new NOAMMetric().getIdentifier(), NOAMMetric.class);
        aMap.put(new FanOutMetric().getIdentifier(), FanOutMetric.class);

        //Register new metrics here in the same format as Above
    }
}

```

The metrics for the program is defined within the function here, inside of the metricAnalysis package. Here users can define their own metrics for use with the program, if they are defined within their own class in the metricAnalysis.Metrics package. This code point is crucial as it acts as the main place for end users to extend the program to suit their required functionality, a key feature discussed with the client.

```

/**
 * Runs the chosen metrics on the chosen input. For now runs Lines of Code
 * metric on all Compilation Units.
 */
public void runMetrics() {
    for(ProjectAST projAst: astHandler.getAstStorage().values()) {
        for(PackageAST packAst: projAst.getPackages().values()) {
            for(ClassAST clasAst: packAst.getClasses().values()) {
                for(MethodAST methAst: clasAst.getMethods().values()) {
                    runMethodLevelMetrics(methAst);
                }
                runClassLevelMetrics(clasAst);
            }
            runPackageLevelMetrics(packAst);
        }
        runProjectLevelMetrics(projAst);
    }
    this.setDone(true);
}

```

The above code snippet shows the hierarchical structure that is iterated over by all metrics over all the nodes inside of the structure displayed in code snippet one. This is the most significant method inside of the program, as it draws together the metrics selected by the user, with the data read in through the JavaParser library, and stores it inside of the AST.

```

public String getResultsAsCSVMethod() {
    String value;
    String csv = "";

    csv = csv.concat("Metric Identifier,Package Identifier,Class Identifier,Method Identifier");
    for (ProjectAST projAST : astHandler.getAstStorage().values()) {
        csv = csv.concat(DELIMITER).concat(projAST.getIdentifier());
    }
    csv = csv.concat("\n");

    for(String identifier : methodMetrics.keySet()) {
        for(String PID: astHandler.getAggregateHeirarchy().keySet()) {
            for (String CID : astHandler.getAggregateHeirarchy().get(PID).keySet()) {
                for (String MID : astHandler.getAggregateHeirarchy().get(PID).get(CID)) {
                    csv = csv.concat(identifier).concat(DELIMITER).concat(PID).concat(DELIMITER).concat(CID).concat(DELIMITER+"\\").concat(MID).concat("\\");
                    for (ProjectAST projAST : astHandler.getAstStorage().values()) {
                        value = null;
                        try {
                            value = projAST.getPackages().get(PID).getClasses().get(CID).getMethods().get(MID).getResults()
                                .get(identifier).getValue().toString();
                        } catch (Exception e) {}
                        if (value == null)
                            csv = csv.concat(DELIMITER);
                        else
                            csv = csv.concat(DELIMITER).concat(value.toString());
                    }
                }
            }
        }
    }
    return csv;
}

```

The final code snippet from the business logic of the program is the way that this data is then accessed and handled by the AST. Displayed above is the way that a CSV file is generated from the data in the AST, by accessing all the layers up to the chosen node, and formatting the data stored, along with identifiers for the metrics performed as well as the node itself.

The final front end user interface was implemented using a very popular Java GUI framework, JavaFX. This framework allows for powerful and diverse GUIs to be developed from a range of styling and design options and implementations. The implementation we chose for this project was a window consisting of multiple scenes, which made up all the different tabs of the program.

```
public static void switchSceneDashboard() throws IOException {
    //sceneAnalysisController.runAnalysis(new ActionEvent());
    if (dashboard == null) {
        URL resource = SceneBuilderMain.class.getResource("Dashboard.fxml");
        FXMLLoader loader = new FXMLLoader(resource);
        dashboard = loader.load();
        sceneDashboardController = loader.getController();
    }
    mainView.setCenter(dashboard);
    sceneDashboardController.updateMetricSelection();
}
```

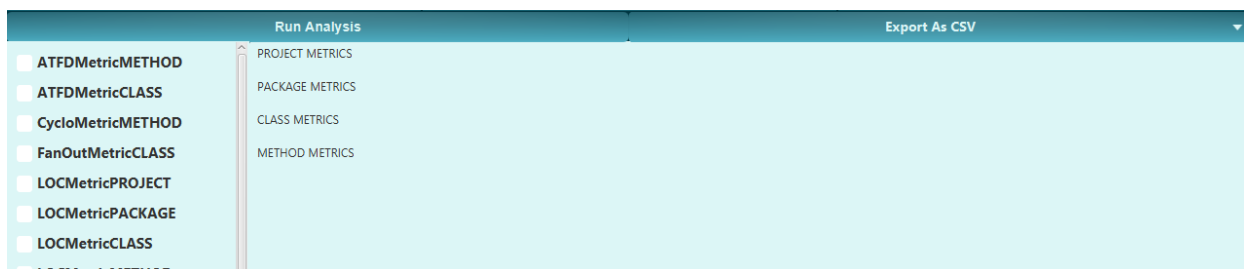
The above code snippet shows how the different tabs were defined, as distinct JavaFX objects known as scenes, where the functionality such as event handlers is described inside a class for the scene itself, and the layout is drawn from a custom XML file that defines the layout and properties of each of the objects that are defined inside of the scene. The entire front end was built using scenes and XML files.

Functionality

As briefly discussed in the project outline section above, the program centres around a user running metric analysis across a program with multiple versions of the same program to compare. The general control flow of the program is discussed above, refer to that for an outline of the way that the program flows, this section of the document will examine each of those UI elements in more detail to explore the way that a user can interact with the system.



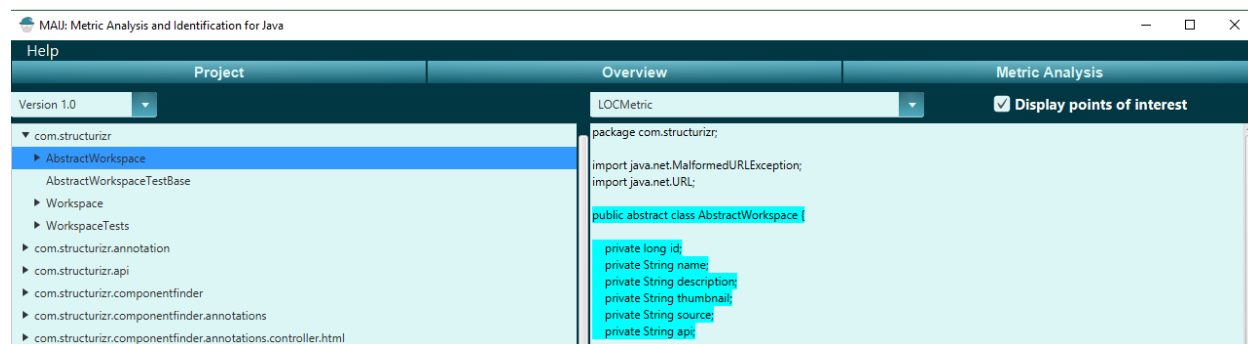
In the front import screen, the user can browse for directories to import as versions to the program. Using the version box users can choose the number of the version they wish to import and can use the browse button to select a directory. Once found, the select button confirms that directory as the selected version, after which a user can select the next version. Once all desired versions are selected, the confirm button imports all the selected data to the program, and the user can continue with the program control flow.



The next applicable window to user in the control flow is the metric analysis window. This is where a user can select the metrics that they wish to apply to the imported data. Users can select as many metrics as they would like from the checkbox menu at the side and when they are ready to perform analysis, they just need to click the run analysis button up the top. This performs the metric analysis on all the data imported. Users can also use this window to export the data as a CSV by clicking the Export as CSV button. This opens a dropdown for the user, where they can select the level of data that they wish to export. They then are presented a file browser to direct the export.

Select Level	Select Metric	
CLASS	LOCMetric	
Identifier	Version 1.0	Version 2.0
com.structurizr.AbstractWorkspace	114.0	161.0
com.structurizr.AbstractWorkspaceTestBase	6.0	7.0
com.structurizr.Workspace	62.0	120.0
com.structurizr.WorkspaceTests	57.0	133.0
com.structurizr.annotation.Component	9.0	9.0
com.structurizr.annotation.UsedByContainer	10.0	10.0

After an analysis the next point of interest for most users will be the overview tab, where users can view the data in a clear presentable way, that makes it easy to browse through the data. At the top users can select the node level that they wish to view data from, from Project, Package, Class or Method. Once a level is selected, the metric list on the other side populates with all the metrics that a user selected to run on that node level in the previous metric analysis section. The table will then be dynamically populated with all the data stored in the AST that is relevant to the selected view. This data is in the same format as the CSV export, for consistency across data analysis applications. Changing the level view or the metric will regenerate the table with the selected data.



If a user wishes for a more in-depth visualisation of the data or how the metric algorithms applied, the project tab gives this visual to the user. In the top left, users can select the version of the program that they wish to view data for. Changing to another version preserves a user's place in the tree, and the metric data being viewed, allowing for easy comparison between versions. The user can then select a node in the tree that they wish to view. The metric drop-down menu will then populate with available metric data available for that node. Once a metric to view has been selected, users can click the display checkbox, and the contribution ranges for the metric will be highlighted in the code view window. Alternating colours are used to differentiate between the different contribution points for a metric and have no meaning other than to distinguish contribution ranges.

Quality and Metrics

As discussed in the above sections, the delivered artefact fulfilled all the user stories and functionality that we agreed upon with client. All agreed user stories fulfilled their acceptance tests that were laid out at the start of semester one.

To ensure the completion and acceptance of all these user stories, and that they are delivered functionally complete, and bug free, a wide range of functional testing was performed on the GUI, testing as many combinations of user interactions as was feasibly possible. This testing was performed multiple times over the end of the implementation phase of the project, and many bugs and problematic user interactions were identified, and modified or fixed to completely fulfil user story acceptance tests.

These testing documents (Appendix 3, 4 ,5) outlined all the user interactions on every UI element in the program and tested as many permutations as could be tested in a reasonable time. All user stories acceptance criteria were assessed, and user stories were accepted or rejected at each stage of the testing based on the findings of the testing documents. The testings continued until all user stories evaluated as passed.

File Browser

Action	Effect	Result
Click Browse	User is prompted with a file browser, to select a directory. Direct file imports are not supported.	PASS
Select a Valid Directory	When a user selects a version of a program that contains code that is <u>parseable</u> by the Java parser library, it is imported to the selected version.	PASS
Select a Non-Valid Directory	When a user tries to import a directory that does not contain valid code, or there is a problem parsing discovered code due to a compilation or syntax error, the user is alerted, and no directory is imported.	FAIL

Fails: If a user tries to import a directory that does not contain valid Java code, an empty version is imported. This should not be permitted, only valid version importation. When a user tries to select a directory that contains unparseable code, it appears that the version has been correctly selected, however when a user confirms import, it will silently fail. Users should be alerted to the failure, and the reason.

Import Screen User Stories:

Version Selection

Acceptance Criteria:

- Can browse computer and select files or directory
- Can name or number the version entered
- Can set the hierarchy or timeline between versions

ACCEPTANCE PASSED

The above section of a testing document shows the way the UI was tested. For every UI element in a window, in this case the popup file browser, the element interactions were tested, and assessed as to whether they passed or failed the criteria we specified for them. When fails were found, they were outlined, with instructions on how to replicate, and the intended behaviour to be fixed. Underneath this

section the user stories related to that element or groups of elements were assessed as to whether the acceptance criteria were accepted, and the user story could be considered complete. Testing ceased once there were no more failures assessed on any UI element, and all user stories were considered accepted.

As discussed in the semester progress section, due to unforeseen time constraints due to the large amount of time required to port the version to the new GUI, this time cut into the time that was allowed for testing on the algorithms that were applied for each of the metrics. It was intended that this would be tested using a large range of automated unit tests, so it could be applied across a large range of data to find any small errors that were present in specific circumstances. The framework was developed for this testing; however, no tests were generated as the time required was too large.

```
class MetricTests {  
  
    SourceImportManager importManager = new SourceImportManager();  
    ASTHandler astHandler;  
    AnalysisHandler analyzer;  
  
    @Before  
    void loadData() {  
        try {  
            importManager.selectImport("./testData");  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
        importManager.addVersion(1.0);  
        astHandler = importManager.confirmImport();  
        analyzer = new AnalysisHandler(astHandler);  
        analyzer.runMetrics();  
        System.out.print(analyzer.getResultsHandler().getResultsAsCSVMethod());  
    }  
}
```

The above code snippet shows the framework that exists already, users need to merely change the selectImport statement to point to whichever data they wish to test on, and then write tests for the metrics.

This automated testing was not performed in favour of manual testing for the metrics. This was performed in a manner where other analysis tools, or manual code checking, were performed on a matching data set and then the results from the external tools and the results from our program algorithms was compared, to validate the correctness of the algorithms.

Manual data checking involved manually checking the code tested to count contribution points for the metric that was being tested and comparing that value against the algorithm value. All metrics were verified correct functionally for the code that was tested, which was a random sample, and is very representative of most of the code. Due to the lack of automated testing however, we cannot verify that the code is functional in 100 percent of cases, as there are many edge cases which were not tested in the manual tests. All the test data from the metric tests can be found in Appendix 6.

Future development

Future extensions of the program were largely defined at the beginning of the project, as user stories that proved too timely or difficult to implement in the time frame available or were passed over to make room for more useful or important user stories. More future extensions were introduced as the scope and target market of the product evolved as client needs and expectations evolved and shifted.

Additional Metrics

The most key extension of the program is the ability for end users or developers to construct and add their own additional metrics to the program, which would seamlessly be recognised and be usable within the program. This has been achieved, and adding additional metrics is only a matter of adding the new metric in a class and adding a single line of code to the program, which allows it to be used by the program. This makes it the ideal choice for extending the program in a non-trivial matter, as there are a vast number of different metrics that can be applied to object-oriented code to gain understanding of the way a program evolves.

Git Importation

One of the most obvious extensions of the project comes in the form of more broad importation options for program versions. Currently only directory importation is supported; however, the next step is to import directly from git repositories hosted online, from Github, Bitbucket or SourceForge. Using the release versions functionality of these repositories, it naturally translates to the versions required for use in the programs. By adding support for remote repository version importing, it greatly streamlines the user control flow of the program, eliminating the need for the user to manually download and catalogue the versions of a program that they wish to analyse.

Target Languages

Another potential extension of this program is the ability to add other target languages to be imported by the program. Whilst much of the framework relies heavily on a library used for the parsing and construction of a Java source tree, implementations of these models exist for other languages, and by changing the backend data structure of the program, other languages could be supported, and would mesh with little effort with the other parts of the framework, which was designed and implemented with a modular focus.

Metric Paradigms

The program currently only contains support for metrics that can be applied to a single method, class or package, and act upon only those single entities. There is potential in the program for implementation of metrics that can operate over multiple versions of a program at one time, e.g. direct comparison metrics, or metrics that operate over multiple classes at one time, e.g. data flow metrics. While the current data structure does not have support for these operations, they could be a potential addition for future users or developers.

Multithreading

As the program performs large amount of algorithm application to many files, this can take a long time when large amounts of data are imported, or many metrics are selected. The program would see great benefit from implementing multithreading in reading in documents and parsing them, in metric analysis application, and from the GUI being on a separate thread from the calculations, making the GUI responsive during analysis operations.

Automated Algorithm Testing

As discussed in the quality section above, there is a framework setup for use with automated unit testing, which creates the data structures required to test the algorithms of the metrics themselves. To implement the automated testing the user would have to design the tests to compare the returned results with a manual analysis or with results from other tools. As the framework already exists within the code base, this would be a small but powerful extension for verifying the correctness of any implemented metrics within the program.

References

IBM. (2018). Advantages of Java. Retrieved from

https://www.ibm.com/support/knowledgecenter/en/ssw_aix_72/com.ibm.aix.performance/advantages_java.htm

JavaParser.org (2018). JavaParser - For processing Java code. Retrieved from <https://javaparser.org/>

Appendix 1 - User Stories

ID	Title	Priority	Points
1	Versions From GitHub	SHOULD	
As a User I want to enter a GitHub project URL and have the program produce the n latest versions of the program from the online repo.			
Acceptance Criteria <ul style="list-style-type: none"> - Detects whether the url matches an online github repo - Lets the user choose n - Gives warning if there aren't enough releases (<2) - Clones the repo and generates up to n latest releases from the repo (or downloads directly) - If less than n releases handles gracefully 			
ID	Title	Priority	Points
2	Versions from BitBucket	COULD	
As a User I want to enter a BitBucket project URL and have the program produce the n latest versions of the program from the online repo.			
Acceptance Criteria <ul style="list-style-type: none"> - Detects whether the url matches an online BitBucket repo - Lets the user choose n - Gives warning if there aren't enough releases <2 - clones the repo and generates up to n latest releases from the repo (or downloads directly) - if less than n releases handles gracefully 			
ID	Title	Priority	Points
3	Versions from SourceForge	COULD	
As a User I want to enter a SourceForge project URL and have the program produce the n latest versions of the program from the online repo.			
Acceptance Criteria <ul style="list-style-type: none"> - Detects whether the url matches an online SourceForge repo - Lets the user choose n - Gives warning if there aren't enough releases <2 - clones the repo and generates up to n latest releases from the repo (or downloads directly) - if less than n releases handles gracefully 			
ID	Title	Priority	Points
4	Enter Versions Manually	MUST	
As a User I want to enter the source code for different versions of the same program manually by browsing my computer.			
Acceptance Criteria <ul style="list-style-type: none"> - Can browse computer and select files or directory - Can name or number the version entered - Can set the heirarchy or timeline between versions 			
Notes			
ID	Title	Priority	Points
5	Version Selection	SHOULD	
As a User I want to select which of the available versions of the program is to be included in the analysis			
Acceptance Criteria <ul style="list-style-type: none"> - A user can select the versions that wish to be analysed, on a case by case basis, with granular control for version exclusions 			
Notes			
ID	Title	Priority	Points
6	Resume Saved project	WONT	
As a User I want to be able to open a saved project to resume a past analysis.			
Acceptance Criteria <ul style="list-style-type: none"> - Session data can be saved and loaded to resume an earlier state 			
Notes <ul style="list-style-type: none"> - Depends on being able to save an analysis/ project (Story 7) 			
ID	Title	Priority	Points
7	Save Analysis/Project	WONT	
As a User I want to save my current Analysis/Project to resume at a later date			
Acceptance Criteria <ul style="list-style-type: none"> - The user can save a session, and in a separate session with the program, load the data and analysis from an earlier session 			
Notes <ul style="list-style-type: none"> - Depends on Story 6 			
ID	Title	Priority	Points
8	Export results	SHOULD	
As a User I want to export the results of my analysis so that I can analyse the data with some other program			
Acceptance Criteria <ul style="list-style-type: none"> - The data from an analysis can be exported into a commonly used file format, for example CSV, and be imported into another program for further data analysis. Potential for support for multiple file formats 			
Notes <ul style="list-style-type: none"> - Probably save all the data as csv 			
ID	Title	Priority	Points
9	Choose Metrics	MUST	
As a user I want to be able to choose which metrics to apply in my analysis			
Acceptance Criteria <ul style="list-style-type: none"> - A user can have granular control over what metrics and analysis are applied to specific data, on a case by case basis, to apply to one version of a program, or an evolution of versions 			
Notes <ul style="list-style-type: none"> - Will depend on the stories that controll how many programs can be analyzed at once and how many metrics can be applied to a program at once 			

ID	Title	Priority	Points
10	Source Code Diff	MUST	
As a User I want to be able to see the source code for two versions side by side with differences highlighted			
Acceptance Criteria			
Two sections of code performing similar or same functionality in different versions of a program can be compared side by side in a visual display window, with markup tools such as highlighting, in order to display the differences between two sections of code			
Notes			
This should apply both to the differences in single files as well as overall structure			

ID	Title	Priority	Points
11	Selective Analysis	SHOULD	
As a user I wish to have control over what aspects and files of the program are analysed - may choose specific classes, files, folders etc and restrict analysis to those			
Acceptance Criteria			
A user can have granular control over what classes or individual methods will be analysed over specific versions of the program			
Notes			

ID	Title	Priority	Points
12	Cause Of Change	MUST	
As a user I want to be able to see the source code that is the source of change in a metric between two versions			
Acceptance Criteria			
A user can view sections of code that caused a change in the value of the metric, a user can iterate between code points that caused the change			
Notes			

ID	Title	Priority	Points
13	New Project	WONT	
As a User I want to be able to create a new analysis project and choose where everything is saved			
Acceptance Criteria			
A user can create a project, and analysis performed lays under this project, which can be saved and resumed later, and a user has control over the locations to which data is saved and stored			

ID	Title	Priority	Points
14	Results Table	MUST	
As a User I want to see the results of the analysis in table format, displaying the programs, versions and values for the metrics, clearly indicating an increase or decrease in a metric across versions of a program			
Acceptance Criteria			
A user can view a table of analysed data, with data clearly displayed, and visual indicators as to the trend of the metrics			

ID	Title	Priority	Points
15	Results Graph	SHOULD	
As a User I want to choose a metric and see a graph of the value across versions of a program			
Acceptance Criteria			
A user can choose a metric to be analysed over various versions of a program, and see a visual graph of the metric over time, and the way it has trended over various versions of the program			

ID	Title	Priority	Points
16	Custom Metrics	MUST	
As a User I want to write custom metrics to use with the software			
Acceptance Criteria			
A user can design a metric class, and have it slot in with other metrics classes, and become a part of the analysis suite. The software should be able to recognise classes that are introduced, and operate on them the same as any other metric class designed with the program			

ID	Title	Priority	Points
17	Annotated Source	COULD	
As a User I want to see the source code annotated with how it affects the value of the selected metric and be able to see what part of the difference between the two versions is responsible for a change in the metric value			
Acceptance Criteria			
As a part of the display of code, with markup and highlights, an annotation can be added on the particular sections that displays to the user the effect that this section had on the metric, and the changes between this version and future or previous versions			

ID	Title	Priority	Points
18	Thorough Documentation	MUST	
As a developer trying to fix bugs or extend the software I want the source code to be very well documented			
Acceptance Criteria			
The code will be well documented, with both comments explaining all non trivial code implementations, as well as standardised documentation tools being implemented for classes and methods, such as javadoc			

ID	Title	Priority	Points
19	Read Java Project hierarchy	MUST	
Can recognize the structure of the project, packages, classes, methods etc.			
Acceptance Criteria			
The program can recognise the structure of the files in a program, and identify classes, methods and packages, and differentiate between them			

ID	Title	Priority	Points
20	Apply metric on function	SHOULD	
Able to apply a metric on a function			
Acceptance Criteria			
The ability for a user to target a specific function or method inside a class to have algorithms performed against			

ID	Title	Priority	Points
21	Apply metric on class	SHOULD	
Able to apply a metric on specific class			
Acceptance Criteria			
The ability for a user to target a specific class for algorithms to be performed against			

ID	Title	Priority	Points
22	Apply metric on package	SHOULD	
Able to apply a metric on a specific package			
Acceptance Criteria			
The ability for the user to specify a package to have the metric algorithm applied against			

ID	Title	Priority	Points
23	Source Code with Syntax Highlighting	MUST	
As a User I want to select Source Code and view it with syntax highlighting			
Acceptance Criteria			
Able to select source code from the imported versions and have it displayed with syntax highlighting			

ID	Title	Priority	Points
24	Apply Metrics across versions	SHOULD	
Able to apply metrics that take multiple versions of a program as input (like edit distance)			
Acceptance Criteria			
Interface accepts metrics that require multiple version of the same code as input			

Appendix 2 - Project Delivery Agreement

Base level System:

User story: As a User I want to enter the source code for different versions of the same program manually by browsing my computer.

Acceptance Criteria:

- Can browse computer and select files or directory
- Can name or number the version entered
- Can set the hierarchy or timeline between versions

Priority: **MUST HAVE**

Delivery: **WILL BE DELIVERED**

User story: As a User I want to write custom metrics to use with the software

Acceptance Criteria: A user can design a metric class, and have it slot in with other metrics classes, and become a part of the analysis suite. The software should be able to recognise classes that are introduced, and operate on them the same as any other metric class designed with the program

Priority: **MUST HAVE**

Delivery: **WILL BE DELIVERED**

User story: Can recognize the structure of the project, packages, classes, methods etc.

Acceptance Criteria: The program can recognise the structure of the files in a program, and identify classes, methods and packages, and differentiate between them

Priority: **MUST HAVE**

Delivery: **WILL BE DELIVERED**

Enhanced System:

User story: As a user I want to be able to the source code that is the source of change in a metric between two versions

Acceptance Criteria: A user can view sections of code that caused a change in the value of the metric, a user can iterate between code points that caused the change

Priority: **MUST HAVE**

Delivery: **WILL BE DELIVERED**

User story: As a user I want to be able to choose which metrics to apply in my analysis

Acceptance Criteria: A user can have granular control over what metrics and analysis are applied to specific data, on a case by case basis, to apply to one version of a program, or an evolution of versions

Priority: **MUST HAVE**

Delivery: **WILL BE DELIVERED**

User story: As a User I want to see the results of the analysis in table format, displaying the programs, versions and values for the metrics, clearly indicating an increase or decrease in a metric across versions of a program

Acceptance Criteria: A user can view a table of analysed data, with data clearly displayed, and visual indicators as to the trend of the metrics

Priority: **MUST HAVE**

Delivery: **WILL BE DELIVERED**

Stretch System:

User story: Able to apply a metric on a function

Acceptance Criteria: The ability for a user to target a specific function or method inside a class to have algorithms performed against

Priority: **SHOULD HAVE**

Delivery: **WILL BE DELIVERED**

User story: Able to apply a metric on specific class

Acceptance Criteria: The ability for a user to target a specific class for algorithms to be performed against

Priority: **SHOULD HAVE**

Delivery: **WILL BE DELIVERED**

User story: Able to apply a metric on a specific package

Acceptance Criteria: The ability for the user to specify a package to have the metric algorithm applied against

Priority: **SHOULD HAVE**

Delivery: **WILL BE DELIVERED**

User story: As a User I want to export the results of my analysis so that I can analyse the data with some other program

Acceptance Criteria: The data from an analysis can be exported into a commonly used file format, for example CSV, and be imported into another program for further data analysis. Potential for support for multiple file formats

Priority: **SHOULD HAVE**

Delivery: **WILL BE DELIVERED**

User story: Able to apply metrics that take multiple versions of a program as input (like edit distance)

Acceptance Criteria: Interface accepts metrics that require multiple version of the same code as input

Priority: **SHOULD HAVE**

Delivery: **NOT DELIVERED**

User story: As a User I want to choose a metric and see a graph of the value across versions of a program

Acceptance Criteria: A user can choose a metric to be analysed over various versions of a program, and see a visual graph of the metric over time, and the way it has trended over various versions of the program

Priority: **SHOULD HAVE**

Delivery: **NOT DELIVERED**

User story: As a user I wish to have control over what aspects and files of the program are analysed - may choose specific classes, files, folders etc and restrict analysis to those

Acceptance Criteria: A user can have granular control over what classes or individual methods will be analysed over specific versions of the program

Priority: **SHOULD HAVE**

Delivery: **NOT DELIVERED**

User story: As a User I want to select Source Code and view it with syntax highlighting

Acceptance Criteria: Able to select source code from the imported versions and have it displayed with syntax highlighting

Priority: **SHOULD HAVE**

Delivery: **NOT DELIVERED**

User story: As a User I want to see the source code for two versions side by side with differences highlighted

Acceptance Criteria: Two sections of code performing similar or same functionality in different versions of a program can be compared side by side in a visual display window, with mark-up tools such as highlighting, to display the differences between two sections of code

Priority: **SHOULD HAVE**

Delivery: **NOT DELIVERED**

User story: As a User I want to select which of the available versions of the program is to be included in the analysis

Acceptance Criteria: A user can select the versions that wish to be analysed, on a case by case basis, with granular control for version exclusions

Priority: **SHOULD HAVE**

Delivery: **NOT DELIVERED**

User story: As a User I want to enter a GitHub project URL and have the program produce the n latest versions of the program from the online repo.

Acceptance Criteria:

- Detects whether the URL matches an online Github repo
- Lets the user choose n
- Gives warning if there aren't enough releases (<2)
- Clones the repo and generates up to n latest releases from the repo (or downloads directly)
- If less than n releases handle gracefully

Priority: **SHOULD HAVE**

Delivery: **NOT DELIVERED**

OUT OF SCOPE/FUTURE PLANS:

User story: As a User I want to enter a BitBucket project URL and have the program produce the n latest versions of the program from the online repo.

Acceptance Criteria:

- Detects whether the URL matches an online Bitbucket repo
- Lets the user choose n
- Gives warning if there aren't enough releases (<2)
- Clones the repo and generates up to n latest releases from the repo (or downloads directly)
- If less than n releases handles gracefully

Priority: **COULD HAVE**

Delivery: **NOT DELIVERED**

User story: As a User I want to enter a SourceForge project URL and have the program produce the n latest versions of the program from the online repo.

Acceptance Criteria:

- Detects whether the URL matches an online SourceForge repo
- Lets the user choose n
- Gives warning if there aren't enough releases (<2)
- Clones the repo and generates up to n latest releases from the repo (or downloads directly)
- If less than n releases handles gracefully

Priority: **COULD HAVE**

Delivery: **NOT DELIVERED**

User story: As a User I want to see the source code annotated with how it affects the value of the selected metric and be able to see what part of the difference between the two versions is responsible for a change in the metric value

Acceptance Criteria: As a part of the display of code, with mark-up and highlights, an annotation can be added on the sections that displays to the user the effect that this section had on the metric, and the changes between this version and future or previous versions

Priority: **COULD HAVE**

Delivery: **NOT DELIVERED**

User story: As a User I want to be able to open a saved project to resume a past analysis.

Acceptance Criteria: Session data can be saved and loaded to resume an earlier state

Priority: **WON'T HAVE**

Delivery: **NOT DELIVERED**

User story: As a User I want to save my current Analysis/Project to resume later

Acceptance Criteria: The user can save a session, and in a separate session with the program, load the data and analysis from an earlier session

Priority: **WON'T HAVE**

Delivery: **NOT DELIVERED**

User story: As a User I want to be able to create a new analysis project and choose where everything is saved

Acceptance Criteria: A user can create a project, and analysis performed lays under this project, which can be saved and resumed later, and a user has control over the locations to which data is saved and stored

Priority: **WON'T HAVE**

Delivery: **NOT DELIVERED**

Appendix 3 – UI Testing 8-10-18

Import Screen

Version Selection

Action	Effect	Result
Clicking up	Users can move up versions and import to that specific version. Users can import as many versions as they wish, no upper limit	PASS
Clicking Down	Users can move down versions and import to the selected number. Users cannot move below version 1.	PASS
Manual number entry	Users can manually enter versions. If a user tries to enter a negative number, or a non-integer, when they press enter to confirm data entry, they are reset to the last selection.	FAIL
Manual number entry	If a string is entered into the version selection area, the user is not alerted that they entered invalid data, and it appears to the user that they have made a valid selection, until they confirm	FAIL

Fails: Manual number entry: When a user tries to enter a non-integer number, with a decimal place, it appears as if this operation is successful, however the imported version will be the last version selected. It should revert to the last selected version when the user presses enter or prompt the user that version numbers must be integers.

File Browser

Action	Effect	Result
Click Browse	User is prompted with a file browser, to select a directory. Direct file imports are not supported.	PASS
Select a Valid Directory	When a user selects a version of a program that contains code that is parseable by the Java parser library, it is imported to the selected version.	PASS
Select a Non-Valid Directory	When a user tries to import a directory that does not contain valid code, or there is a problem parsing discovered code due to a compilation or syntax error, the user is alerted, and no directory is imported.	FAIL

Fails: If a user tries to import a directory that does not contain valid Java code, an empty version is imported. This should not be permitted, only valid version importation. When a user tries to select a directory that contains unparseable code, it appears that the version has been correctly selected, however when a user confirms import, it will silently fail. Users should be alerted to the failure, and the reason.

Import Screen User Stories:

Version Selection

Acceptance Criteria:

- Can browse computer and select files or directory
- Can name or number the version entered
- Can set the hierarchy or timeline between versions

ACCEPTANCE PASSED

Structure Recognition

Acceptance Criteria:

The program can recognise the structure of the files in a program, and identify classes, methods and packages, and differentiate between them.

ACCEPTANCE PASSED

Project Window

Version Selection

Action	Effect	Result
Select A Version	Users can select a version to view the structure of. Versions are generated from the versions imported into the program.	PASS
Version Switching	Clicking on a version switches to that version, preserving the users place in the structure in the tree.	FAIL

Fails: When user switch between versions of a program, the place in the tree is lost. Minor problem.

Version Select User Stories:

Structure Traversal

Action	Effect	Result
Click on a Package	If the package has metadata parsed by the javaparser, such as a package contents description, it is displayed to the right in the code window, otherwise the code window remains blank.	PASS
Expand a Package	When a user clicks the arrow next to a package, or double clicks the package name, the package is expanded to show all the contained Java classes within the package. Double clicking again collapses the package.	PASS
Click on a Package	When a user clicks on a class, the contents of that class are displayed in the code window, as read by the Javaparser.	PASS
Expand a Class	When a user clicks the arrow next to a class, or double clicks the package name, the package is expanded to show all the contained Java methods within the class. Double clicking again collapses the class.	PASS
Click on a Method	When a user clicks on a method, the class is shown in the code window, with that method highlighted. If the method is off the screen in the code window, the user is taken to that part of the class and displayed the method.	PASS

Fails: -

Structure Traversal User Stories:

Code View

Action	Effect	Result
Highlight from the Window	Users can select and highlight code.	PASS
Interact with the window	Users should not be able to type or edit code in the window, as it is for display only, not code editing.	FAIL

Fails: Users should not be able to edit code inside the code view, or interact with the window, as it may mislead users into thinking they are making changes that are not preserved.

Code View User Stories

Code Contribution

Acceptance Criteria: A user can view sections of code that caused a change in the value of the metric, a user can iterate between code points that caused the change.

ACCEPTANCE FAILED

Overview Window

Level selection

Action	Effect	Result
Select Project	Sets the metric selection menu to available project level metrics.	PASS
Select Package	Sets the metric selection menu to available package level metrics.	PASS
Select Class	Sets the metric selection menu to available class level metrics.	PASS
Select Method	Sets the metric selection menu to available method level metrics.	PASS
Switch Level	On any level switch the populated data changes to match the selected level and metric	

Fails: -

Level Selection User Stories:

Metric Package Application

Acceptance Criteria: The ability for the user to specify a Package to have the metric algorithm applied against.

ACCEPTANCE PASSED

Metric Class Application

Acceptance Criteria: The ability for the user to specify a Class to have the metric algorithm applied against.

ACCEPTANCE PASSED

Metric Method Application

Acceptance Criteria: The ability for the user to specify a Method to have the metric algorithm applied against.

ACCEPTANCE PASSED

Metric Selection

Action	Effect	Result
No Analysis Run	The list of metrics is empty no analysis has been performed, and no metrics have been applied.	FAIL
Analysis Run	The list of available metrics matches the level specification selected, populated with available metric data.	PASS
Switch Metric	Populates the results table with data associated with the selected level and metric.	PASS

Fails: When No analysis has been performed, the metric selection checkbox should read no analysis performed, or something similar, so users are aware of the control flow of the application.

Metric Selection User Stories:

Results Table

Action	Effect	Result
Loading Data	The table populates with the data from the metric analysis.	PASS
Clicking Column Headers	The table will reorganise itself based on ascending, descending order, or back to the default ordering, based on which column it is ordered by. Clicking through all the options cycles back to the start.	PASS
Dynamic Column Generation	Columns are generated based on the number of versions imported to the program	PASS
Dynamic Scroll Generation	Scroll bars are dynamically generated for both vertical and horizontal scroll, only when required, keeping the interface clean	PASS
Null indicator	There is a clear indicator to the user when there is no data for a datapoint	PASS

Fails: -

Results Table User Stories:

Results Table:

Acceptance Criteria: A user can view a table of analysed data, with data clearly displayed, and visual indicators as to the trend of the metrics.

ACCEPTANCE PASSED

Metric Analysis Page

Run Analysis

Action	Effect	Result
Run Analysis	The analysis with the selected metrics is performed against all versions, the result is displayed to the user in the code window, in raw data dump data form	PASS
Run Analysis, No Metrics Selected	If no metrics are selected, the user should be alerted that they must select at least one metric for analysis to be performed.	FAIL

Fails: When a user tries to run an analysis with no metric selected, it silently fails. The user should be alerted as to why no analysis is running.

Run Analysis User Stories:

Metric Selection Boxes

Action	Effect	Result
No Metrics by default	As it is more likely that a user may want to run a specific metric, or subset of metrics, the default is to have no metrics selected, so users can specify only the ones they want.	PASS
Check a Box	Checking a box adds that metric to the list of applied metrics.	PASS
Uncheck a Box	Unchecking a box removes that metrics from the list of applied metrics.	PASS
Automatic Scaling	Metric checkboxes are drawn from the metrics that are within the program, and automatically add any additional metrics added to the program.	PASS
Automatic Scrolling	If too many metrics are added to fit onscreen at one time, a scroll box is added to allow all metrics to be viewed.	FAIL

Fails: When too many metrics are added to the program currently, they will list off the screen, and the user will be unable to click on them to add them.

Metric Selection User Stories:

Custom Metrics

Acceptance Criteria: A user can design a metric class, and have it slot in with other metrics classes, and become a part of the analysis suite. The software should be able to recognise classes that are introduced and operate on them the same as any other metric class designed with the program.

ACCEPTANCE PASSED

Metric Selection

Acceptance Criteria: A user can view sections of code that caused a change in the value of the metric, a user can iterate between code points that caused the change.

ACCEPTANCE PASSED

Export as CSV

Action	Effect	Result
Click on the Dropdown Menu	The menu displayed contains all the export options that are available to the user.	PASS
Select Method Metrics	The text window should populate with the CSV string data, and the user should be presented with a dialog box to select a location to save the data into, in CSV format.	FAIL
Select Class Metrics	The text window should populate with the CSV string data, and the user should be presented with a dialog box to select a location to save the data into, in CSV format.	FAIL
Select Package Metrics	The text window should populate with the CSV string data, and the user should be presented with a dialog box to select a location to save the data into, in CSV format.	FAIL

Fails: Currently when selected, the text window is populated with the data, but the user is not presented with a way to save the data, only copy and pasting it into a separate file. The user needs to be able to select a location and a name to save the data to.

Export as CSV User Stories

Acceptance Criteria: The data from an analysis can be exported into a commonly used file format, for example CSV, and be imported into another program for further data analysis. Potential for support for multiple file formats.

ACCEPTANCE FAILED

Menu Bar

File Bar

Action	Effect	Result
Click on the Dropdown Menu	A menu is presented to the user with associated options.	PASS
Click Close	The Program should ask for confirmation before closing.	FAIL

Fails: The close button currently does nothing. Should be implemented.

Edit Bar

Action	Effect	Result
Click on the Dropdown Menu	A menu is presented to the user with associated options.	PASS
Click Delete		FAIL

Fails: The delete button currently does nothing. Should be removed.

Help Bar

Action	Effect	Result
Click on the Dropdown Menu	A menu is presented to the user with associated options.	PASS
Click About	The user should be presented with a dialog displaying information about the program, as well as a short usage guide.	FAIL

Fails: The about button currently does nothing. Should be implemented.

Appendix 4 - UI Testing 15-10-18

Import Screen

Version Selection

Action	Effect	Result
Clicking up	Users can move up versions and import to that specific version. Users can import as many versions as they wish, no upper limit	PASS
Clicking Down	Users can move down versions and import to the selected number. Users cannot move below version 1.	PASS

Fails:

File Browser

Action	Effect	Result
Click Browse	User is prompted with a file browser, to select a directory. Direct file imports are not supported.	PASS
Select a Valid Directory	When a user selects a version of a program that contains code that is parseable by the Java parser library, it is imported to the selected version.	PASS
Select an Empty Directory	When a user selects a directory that contains no valid .java files, the user is alerted to the empty directory, and nothing is imported into the program.	PASS
Select an Error Directory	When a user imports a directory that contains non-valid code, the file that contains non-valid code, the directory is imported, with broken files being ignored by the java parser.	FAIL

Fails: Users should be alerted that not all code read into the program correctly and be shown which files failed to import.

Import Screen User Stories:

Version Selection

Acceptance Criteria:

- Can browse computer and select files or directory
- Can name or number the version entered
- Can set the hierarchy or timeline between versions

ACCEPTANCE PASSED

Structure Recognition

Acceptance Criteria:

The program can recognise the structure of the files in a program, and identify classes, methods and packages, and differentiate between them.

ACCEPTANCE PASSED

Project Window

Version Selection

Action	Effect	Result
Select A Version	Users can select a version to view the structure of. Versions are generated from the versions imported into the program.	PASS
Version Switching	Clicking on a version switches to that version, preserving the users place in the structure in the tree.	PASS

Fails:

Version Select User Stories:

Structure Traversal

Action	Effect	Result
Click on a Package	If the package has metadata parsed by the javaparser, such as a package contents description, it is displayed to the right in the code window, otherwise the code window remains blank.	PASS
Expand a Package	When a user clicks the arrow next to a package, or double clicks the package name, the package is expanded to show all the contained Java classes within the package. Double clicking again collapses the package.	PASS
Click on a Package	When a user clicks on a class, the contents of that class are displayed in the code window, as read by the Javaparser.	PASS
Expand a Class	When a user clicks the arrow next to a class, or double clicks the package name, the package is expanded to show all the contained Java methods within the class. Double clicking again collapses the class.	PASS
Click on a Method	When a user clicks on a method, the class is shown in the code window, with that method highlighted. If the method is off the screen in the code window, the user is taken to that part of the class and displayed the method.	PASS

Fails:

Structure Traversal User Stories:

Code View

Action	Effect	Result
Highlight from the Window	Users can select and highlight code.	PASS
Interact with the window	Users should not be able to type or edit code in the window, as it is for display only, not code editing.	PASS

Fails:

Points of Interest

Action	Effect	Result
Dropdown menu	Dropdown menu populates with available metrics for the selected AST element on the left.	PASS
Selecting a metric with checkbox checked	The ranges for the selected metric are displayed on the screen.	PASS
Selecting a metric with the checkbox unchecked	The metric is selected, with no ranges shown.	PASS
Checking the checkbox with a metric selected.	The selected metric ranges are highlighted in the code window.	PASS
Unchecking the checkbox with a metric selected	The selected metric ranges are removed from the window	FAIL
Checking Points of Interest Display	The points of interest for the chosen metric are highlighted in the code view window.	FAIL

Fails: When highlight ranges are applied and the user unchecks the display button, the highlights should disappear, they currently remain. The ranges for metrics are displayed incorrectly, positioning incorrect.

Points of Interest User Stories

Code Contribution

Acceptance Criteria: A user can view sections of code that caused a change in the value of the metric, a user can iterate between code points that caused the change.

ACCEPTANCE PASSED

Overview Window

Level selection

Action	Effect	Result
Select Project	Sets the metric selection menu to available project level metrics.	PASS
Select Package	Sets the metric selection menu to available package level metrics.	PASS
Select Class	Sets the metric selection menu to available class level metrics.	PASS
Select Method	Sets the metric selection menu to available method level metrics.	PASS
Switch Level	On any level switch the populated data changes to match the selected level and metric	PASS

Fails: -

Level Selection User Stories:

Metric Package Application

Acceptance Criteria: The ability for the user to specify a Package to have the metric algorithm applied against.

ACCEPTANCE PASSED

Metric Class Application

Acceptance Criteria: The ability for the user to specify a Class to have the metric algorithm applied against.

ACCEPTANCE PASSED

Metric Method Application

Acceptance Criteria: The ability for the user to specify a Method to have the metric algorithm applied against.

ACCEPTANCE PASSED

Metric Selection

Action	Effect	Result
No Analysis Run	The list of metrics is empty no analysis has been performed, and no metrics have been applied.	FAIL
Analysis Run	The list of available metrics matches the level specification selected, populated with available metric data.	PASS
Switch Metric	Populates the results table with data associated with the selected level and metric.	PASS

Fails: When No analysis has been performed, the metric selection checkbox should read no analysis performed, or something similar, so users are aware of the control flow of the application.

Metric Selection User Stories:

Results Table

Action	Effect	Result
Loading Data	The table populates with the data from the metric analysis.	PASS
Clicking Column Headers	The table will reorganise itself based on ascending, descending order, or back to the default ordering, based on which column it is ordered by. Clicking through all the options cycles back to the start.	PASS
Dynamic Column Generation	Columns are generated based on the number of versions imported to the program	PASS
Dynamic Scroll Generation	Scroll bars are dynamically generated for both vertical and horizontal scroll, only when required, keeping the interface clean	PASS
Null indicator	There is a clear indicator to the user when there is no data for a datapoint	PASS

Fails: -

Results Table User Stories:

Results Table:

Acceptance Criteria: A user can view a table of analysed data, with data clearly displayed, and visual indicators as to the trend of the metrics.

ACCEPTANCE PASSED

Metric Analysis Page

Run Analysis

Action	Effect	Result
Run Analysis	The analysis with the selected metrics is performed against all versions, the result is displayed to the user in the code window, in raw data dump data form	PASS
Run Analysis, No Metrics Selected	If no metrics are selected, the user should be alerted that they must select at least one metric for analysis to be performed.	PASS

Fails:

Run Analysis User Stories:

Metric Selection Boxes

Action	Effect	Result
No Metrics by default	As it is more likely that a user may want to run a specific metric, or subset of metrics, the default is to have no metrics selected, so users can specify only the ones they want.	PASS
Check a Box	Checking a box adds that metric to the list of applied metrics.	PASS
Uncheck a Box	Unchecking a box removes that metrics from the list of applied metrics.	PASS
Automatic Scaling	Metric checkboxes are drawn from the metrics that are within the program, and automatically add any additional metrics added to the program.	PASS

Automatic Scrolling	If too many metrics are added to fit onscreen at one time, a scroll box is added to allow all metrics to be viewed.	PASS
---------------------	---	------

Fails:

Metric Selection User Stories:

Custom Metrics

Acceptance Criteria: A user can design a metric class, and have it slot in with other metrics classes, and become a part of the analysis suite. The software should be able to recognise classes that are introduced and operate on them the same as any other metric class designed with the program.

ACCEPTANCE PASSED

Metric Selection

Acceptance Criteria: A user can view sections of code that caused a change in the value of the metric, a user can iterate between code points that caused the change.

ACCEPTANCE PASSED

Export as CSV

Action	Effect	Result
Click on the Dropdown Menu	The menu displayed contains all the export options that are available to the user.	PASS
Select Method Metrics	The text window should populate with the CSV string data, and the user should be presented with a dialog box to select a location to save the data into, in CSV format.	FAIL
Select Class Metrics	The text window should populate with the CSV string data, and the user should be presented with a dialog box to select a location to save the data into, in CSV format.	FAIL
Select Package Metrics	The text window should populate with the CSV string data, and the user should be presented with a dialog box to select a location to save the data into, in CSV format.	FAIL

Fails: Currently when selected, the text window is populated with the data, but the user is not presented with a way to save the data, only copy and pasting it into a separate file. The user needs to be able to select a location and a name to save the data to.

Export as CSV User Stories

Acceptance Criteria: The data from an analysis can be exported into a commonly used file format, for example CSV, and be imported into another program for further data analysis. Potential for support for multiple file formats.

ACCEPTANCE FAILED

Menu Bar

File Bar

Action	Effect	Result
Click on the Dropdown Menu	A menu is presented to the user with associated options.	PASS
Click Close	The Program should ask for confirmation before closing.	FAIL

Fails: The close button currently does nothing. Should be implemented.

Edit Bar

Action	Effect	Result
Click on the Dropdown Menu	A menu is presented to the user with associated options.	PASS
Click Delete		FAIL

Fails: The delete button currently does nothing. Should be removed.

Help Bar

Action	Effect	Result
Click on the Dropdown Menu	A menu is presented to the user with associated options.	PASS
Click About	The user should be presented with a dialog displaying information about the program, as well as a short usage guide.	FAIL

Fails: The about button currently does nothing. Should be implemented.

Appendix 5 - UI Testing 21-10-18

Import Screen

Version Selection

Action	Effect	Result
Clicking up	Users can move up versions and import to that specific version. Users can import as many versions as they wish, no upper limit	PASS
Clicking Down	Users can move down versions and import to the selected number. Users cannot move below version 1.	PASS

Fails:

File Browser

Action	Effect	Result
Click Browse	User is prompted with a file browser, to select a directory. Direct file imports are not supported.	PASS
Select a Valid Directory	When a user selects a version of a program that contains code that is parseable by the Java parser library, it is imported to the selected version.	PASS
Select an Empty Directory	When a user selects a directory that contains no valid .java files, the user is alerted to the empty directory, and nothing is imported into the program.	PASS
Select an Error Directory	When a user imports a directory that contains non-valid code, the file that contains non-valid code, the directory is imported, with broken files being ignored by the java parser.	PASS

Fails:

Import Screen User Stories:

Version Selection

Acceptance Criteria:

- Can browse computer and select files or directory
- Can name or number the version entered
- Can set the hierarchy or timeline between versions

ACCEPTANCE PASSED

Structure Recognition

Acceptance Criteria:

The program can recognise the structure of the files in a program, and identify classes, methods and packages, and differentiate between them.

ACCEPTANCE PASSED

Project Window

Version Selection

Action	Effect	Result
Select A Version	Users can select a version to view the structure of. Versions are generated from the versions imported into the program.	PASS
Version Switching	Clicking on a version switches to that version, preserving the users place in the structure in the tree.	PASS

Fails:

Version Select User Stories:

Structure Traversal

Action	Effect	Result
Click on a Package	If the package has metadata parsed by the javaparser, such as a package contents description, it is displayed to the right in the code window, otherwise the code window remains blank.	PASS
Expand a Package	When a user clicks the arrow next to a package, or double clicks the package name, the package is expanded to show all the contained Java classes within the package. Double clicking again collapses the package.	PASS
Click on a Package	When a user clicks on a class, the contents of that class are displayed in the code window, as read by the Javaparser.	PASS
Expand a Class	When a user clicks the arrow next to a class, or double clicks the package name, the package is expanded to show all the contained Java methods within the class. Double clicking again collapses the class.	PASS
Click on a Method	When a user clicks on a method, the class is shown in the code window, with that method highlighted. If the method is off the screen in the code window, the user is taken to that part of the class and displayed the method.	PASS

Fails:

Structure Traversal User Stories:

Code View

Action	Effect	Result
Highlight from the Window	Users can select and highlight code.	PASS
Interact with the window	Users should not be able to type or edit code in the window, as it is for display only, not code editing.	PASS

Fails:

Points of Interest

Action	Effect	Result
Dropdown menu	Dropdown menu populates with available metrics for the selected AST element on the left.	PASS
Selecting a metric with checkbox checked	The ranges for the selected metric are displayed on the screen.	PASS
Selecting a metric with the checkbox unchecked	The metric is selected, with no ranges shown.	PASS
Checking the checkbox with a metric selected.	The selected metric ranges are highlighted in the code window.	PASS
Unchecking the checkbox with a metric selected	The selected metric ranges are removed from the window	PASS
Checking Points of Interest Display	The points of interest for the chosen metric are highlighted in the code view window.	PASS

Fails:

Points of Interest User Stories

Code Contribution

Acceptance Criteria: A user can view sections of code that caused a change in the value of the metric, a user can iterate between code points that caused the change.

ACCEPTANCE PASSED

Overview Window

Level selection

Action	Effect	Result
Select Project	Sets the metric selection menu to available project level metrics.	PASS
Select Package	Sets the metric selection menu to available package level metrics.	PASS
Select Class	Sets the metric selection menu to available class level metrics.	PASS
Select Method	Sets the metric selection menu to available method level metrics.	PASS
Switch Level	On any level switch the populated data changes to match the selected level and metric	PASS

Fails: -

Level Selection User Stories:

Metric Package Application

Acceptance Criteria: The ability for the user to specify a Package to have the metric algorithm applied against.

ACCEPTANCE PASSED

Metric Class Application

Acceptance Criteria: The ability for the user to specify a Class to have the metric algorithm applied against.

ACCEPTANCE PASSED

Metric Method Application

Acceptance Criteria: The ability for the user to specify a Method to have the metric algorithm applied against.

ACCEPTANCE PASSED

Metric Selection

Action	Effect	Result
No Analysis Run	The user cannot open the overview tab, as no data exists for it.	PASS
Analysis Run	The list of available metrics matches the level specification selected, populated with available metric data.	PASS
Switch Metric	Populates the results table with data associated with the selected level and metric.	PASS

Fails:

Metric Selection User Stories:

Results Table

Action	Effect	Result
Loading Data	The table populates with the data from the metric analysis.	PASS
Clicking Column Headers	The table will reorganise itself based on ascending, descending order, or back to the default ordering, based on which column it is ordered by. Clicking through all the options cycles back to the start.	PASS
Dynamic Column Generation	Columns are generated based on the number of versions imported to the program	PASS
Dynamic Scroll Generation	Scroll bars are dynamically generated for both vertical and horizontal scroll, only when required, keeping the interface clean	PASS
Null indicator	There is a clear indicator to the user when there is no data for a datapoint	PASS

Fails: -

Results Table User Stories:

Results Table:

Acceptance Criteria: A user can view a table of analysed data, with data clearly displayed, and visual indicators as to the trend of the metrics.

ACCEPTANCE PASSED

Metric Analysis Page

Run Analysis

Action	Effect	Result
Run Analysis	The analysis with the selected metrics is performed against all versions, the result is displayed to the user in the code window, in raw data dump data form	PASS
Run Analysis, No Metrics Selected	If no metrics are selected, the user should be alerted that they must select at least one metric for analysis to be performed.	PASS

Fails:

Run Analysis User Stories:

Metric Selection Boxes

Action	Effect	Result
No Metrics by default	As it is more likely that a user may want to run a specific metric, or subset of metrics, the default is to have no metrics selected, so users can specify only the ones they want.	PASS
Check a Box	Checking a box adds that metric to the list of applied metrics.	PASS
Uncheck a Box	Unchecking a box removes that metrics from the list of applied metrics.	PASS
Automatic Scaling	Metric checkboxes are drawn from the metrics that are within the program, and automatically add any additional metrics added to the program.	PASS
Automatic Scrolling	If too many metrics are added to fit onscreen at one time, a scroll box is added to allow all metrics to be viewed.	PASS

Fails:

Metric Selection User Stories:

Custom Metrics

Acceptance Criteria: A user can design a metric class, and have it slot in with other metrics classes, and become a part of the analysis suite. The software should be able to recognise classes that are introduced and operate on them the same as any other metric class designed with the program.

ACCEPTANCE PASSED

Metric Selection

Acceptance Criteria: A user can view sections of code that caused a change in the value of the metric, a user can iterate between code points that caused the change.

ACCEPTANCE PASSED

Export as CSV

Action	Effect	Result
Click on the Dropdown Menu	The menu displayed contains all the export options that are available to the user.	PASS
Select Method Metrics	The text window should populate with the CSV string data, and the user should be presented with a dialog box to select a location to save the data into, in CSV format.	PASS
Select Class Metrics	The text window should populate with the CSV string data, and the user should be presented with a dialog box to select a location to save the data into, in CSV format.	PASS
Select Package Metrics	The text window should populate with the CSV string data, and the user should be presented with a dialog box to select a location to save the data into, in CSV format.	PASS

Fails:

Export as CSV User Stories

Acceptance Criteria: The data from an analysis can be exported into a commonly used file format, for example CSV, and be imported into another program for further data analysis. Potential for support for multiple file formats.

ACCEPTANCE FAILED

Menu Bar

Help Bar

Action	Effect	Result
Click on the Dropdown Menu	A menu is presented to the user with associated options.	PASS
Click About	The user should be presented with a dialog displaying information about the program, as well as a short usage guide.	PASS

Fails:

Appendix 6 - Metric Functionality Testing

Manual Data Analysis

To test the functionality of these metrics, a Java class that contained contribution points for all the applicable metrics was manually analysed, and the found results were compared against the data generated by MAIJ, to verify the correctness of the metrics. The code analysed was pulled from a public Github project, which will be linked below this section, as well as the full class that was analysed.

Class Metrics

LOCMetric: This metric is a measure of the lines of code inside a class. This is measured by every line of code that contributes to the body of a class, e.g. anything inside the class declaration. This excludes import statements, as they do not form part of the class declaration in Java.

Manual Data: 108

Program Generated Data: 108

MATCH

NOSMetric: This metric is a measure of the number of statements inside of a class or method. This is measured by every assignment statement or return statement. This excludes variable declarations without a data assignment.

Manual Data: 12

Program Generated Data: 12

MATCH

NOAMMetric: This metric is a measure of number of accessor methods contained within a class, e.g. any method that provides access to, or returns, internal class data, whether that data is public, protected or private, and accessible through direct indexing.

Manual Data: 10

Program Generated Data: 10

MATCH

FanOutMetric: This metric is a measure of how many non-primitive data types are introduced or declared inside of a class or a method. In Java, this is any internal data type that starts with a capital letter as opposed to lowercase, e.g. "int" vs "Integer", or any imported data types or structures.

Manual Data: 1

Program Generated Data: 1

MATCH

NOPAMetric: This metric is a measure of the amount of public accessor methods contained within a class. This differs from the NOAMMetric, as it contains only public accessor methods, so any protected or private methods used for internal data manipulation, such as singleton instance generators.

Manual Data: 10

Program Generated Data: 10

MATCH

ATFDMetric: This metric is a measure of access to foreign data, e.g. all data that can be accessed from outside of the class. This is contributed to by any access through direct indexing to a field, or through an accessor method, for any class or method declared out of the scope of the class.

Manual Data: 0

Program Generated Data: 0

MATCH

Project Link

<https://github.com/structurizr/java/releases/tag/1.0.0-RC5>

Code found in Code Excerpt 1

Method Metrics

The method level metrics were tested on a different method to the class level metrics, a large method inside of a library from google used for pattern matching. The large methods inside of this class made it ideal for evaluating the class and method level metrics.

CycloMetric: This metric is a measure of cyclomatic complexity of a method, the number of different traversal paths that exist within a code block. This increments every time there is a choice to be made, such as an “if” statement or a “switch” statement.

Used Method: diff_CleanupMerge(LinkedList)

Manual Data: 34

Program Generated Data: 34

MATCH

MaxNestingMetric: This metric is a measure of maximum nesting of code paths contained within a method. It increments every time there is a statement nested within another statement, either through a choice or through a loop.

Used Method: `diff_CleanupMerge(LinkedList)`

Manual Data: 6

Program Generated Data: 6

MATCH

Project Link

<https://github.com/google/diff-match-patch>

Code found in Code Excerpt 2

Code Excerpt 1

Relative Path:

\structurizr-core\src\com\structurizr\AbstractWorkspace.java

```
package com.structurizr;

/**
 * The superclass for regular and encrypted workspaces.
 */
public abstract class AbstractWorkspace {

    private long id;
    private String name;
    private String description;
    private String version;
    private String thumbnail;

    protected AbstractWorkspace() {
    }

    AbstractWorkspace(String name, String description) {
        this.name = name;
        this.description = description;
    }

    /**
     * Gets the ID of this workspace.
     *
     * @return the ID (a positive integer)
     */
    public long getId() {
        return this.id;
    }

    /**
     * Sets the ID of this workspace.
     *
     * @param id the ID (a positive integer)
     */
    public void setId(long id) {
        this.id = id;
    }

    /**
     * Gets the name of this workspace.
     *
     * @return the name, as a String
     */
    public String getName() {
        return name;
    }
}
```

```
/**
 * Sets the name of this workspace.
 *
 * @param name      the name, as a String
 */
public void setName(String name) {
    this.name = name;
}

/**
 * Gets the description of this workspace.
 *
 * @return the description, as a String
 */
public String getDescription() {
    return description;
}

/**
 * Sets the description of this workspace.
 *
 * @param description the description, as a String
 */
public void setDescription(String description) {
    this.description = description;
}

/**
 * Gets the version of this workspace.
 *
 * @return the version, as a String
 */
public String getVersion() {
    return version;
}

/**
 * Sets the version of this workspace.
 *
 * @param version the version, as a String (e.g. 1.0.1, a git hash, etc).
 */
public void setVersion(String version) {
    this.version = version;
}

/**
 * Gets the thumbnail associated with this workspace.
 *
 * @return a Base64 encoded PNG file as a Data URI (data:image/png;base64)
 *         or null if there is no thumbnail
 */
public String getThumbnail() {
    return thumbnail;
}
```

```

/**
 * Sets the thumbnail associated with this workspace.
 *
 * @param thumbnail      a Base64 encoded PNG file as a Data URI (data:image/png;base64)
 */
public void setThumbnail(String thumbnail) {
    this.thumbnail = thumbnail;
}
}

```

Code Excerpt 2

Relative Path

/java/src/name/name/fraser/neil/plaintext/diff_metch_patch.java

```

public void diff_cleanupMerge(LinkedList<Diff> diffs) {
    diffs.add(new Diff(Operation.EQUAL, "")); // Add a dummy entry at the end.
    ListIterator<Diff> pointer = diffs.listIterator();
    int count_delete = 0;
    int count_insert = 0;
    String text_delete = "";
    String text_insert = "";
    Diff thisDiff = pointer.next();
    Diff prevEqual = null;
    int commonlength;
    while (thisDiff != null) {
        switch (thisDiff.operation) {
            case INSERT:
                count_insert++;
                text_insert += thisDiff.text;
                prevEqual = null;
                break;
            case DELETE:
                count_delete++;
                text_delete += thisDiff.text;
                prevEqual = null;
                break;
            case EQUAL:
                if (count_delete + count_insert > 1) {
                    boolean both_types = count_delete != 0 && count_insert != 0;
                    // Delete the offending records.
                    pointer.previous(); // Reverse direction.
                    while (count_delete-- > 0) {
                        pointer.previous();
                        pointer.remove();
                    }
                    while (count_insert-- > 0) {
                        pointer.previous();
                        pointer.remove();
                    }
                }
                if (both_types) {
                    // Factor out any common prefixies.
                    commonlength = diff_commonPrefix(text_insert, text_delete);
                    if (commonlength != 0) {
                        if (pointer.hasPrevious()) {
                            thisDiff = pointer.previous();
                            assert thisDiff.operation == Operation.EQUAL
                                : "Previous diff should have been an equality.";
                            thisDiff.text += text_insert.substring(0, commonlength);
                            pointer.next();
                        } else {
                            pointer.add(new Diff(Operation.EQUAL,
                                text_insert.substring(0, commonlength)));
                        }
                    }
                }
            }
        }
    }
}

```

```

        text_insert = text_insert.substring(commonlength);
        text_delete = text_delete.substring(commonlength);
    }
    // Factor out any common suffixies.
    commonlength = diff_commonSuffix(text_insert, text_delete);
    if (commonlength != 0) {
        thisDiff = pointer.next();
        thisDiff.text = text_insert.substring(text_insert.length()
            - commonlength) + thisDiff.text;
        text_insert = text_insert.substring(0, text_insert.length()
            - commonlength);
        text_delete = text_delete.substring(0, text_delete.length()
            - commonlength);
        pointer.previous();
    }
}
// Insert the merged records.
if (text_delete.length() != 0) {
    pointer.add(new Diff(Operation.DELETE, text_delete));
}
if (text_insert.length() != 0) {
    pointer.add(new Diff(Operation.INSERT, text_insert));
}
// Step forward to the equality.
thisDiff = pointer.hasNext() ? pointer.next() : null;
} else if (prevEqual != null) {
    // Merge this equality with the previous one.
    prevEqual.text += thisDiff.text;
    pointer.remove();
    thisDiff = pointer.previous();
    pointer.next(); // Forward direction
}
count_insert = 0;
count_delete = 0;
text_delete = "";
text_insert = "";
prevEqual = thisDiff;
break;
}
thisDiff = pointer.hasNext() ? pointer.next() : null;
}
if (diffs.getLast().text.length() == 0) {
    diffs.removeLast(); // Remove the dummy entry at the end.
}

/*
 * Second pass: look for single edits surrounded on both sides by equalities
 * which can be shifted sideways to eliminate an equality.
 * e.g: A<ins>BA</ins>C -> <ins>AB</ins>AC
 */

```

```

boolean changes = false;
// Create a new iterator at the start.
// (As opposed to walking the current one back.)
pointer = diffs.listIterator();
Diff prevDiff = pointer.hasNext() ? pointer.next() : null;
thisDiff = pointer.hasNext() ? pointer.next() : null;
Diff nextDiff = pointer.hasNext() ? pointer.next() : null;
// Intentionally ignore the first and last element (don't need checking).
while (nextDiff != null) {
    if (prevDiff.operation == Operation.EQUAL &&
        nextDiff.operation == Operation.EQUAL) {
        // This is a single edit surrounded by equalities.
        if (thisDiff.text.endsWith(prevDiff.text)) {
            // Shift the edit over the previous equality.
            thisDiff.text = prevDiff.text
                + thisDiff.text.substring(0, thisDiff.text.length()
                    - prevDiff.text.length());
            nextDiff.text = prevDiff.text + nextDiff.text;
            pointer.previous(); // Walk past nextDiff.
            pointer.previous(); // Walk past thisDiff.
            pointer.previous(); // Walk past prevDiff.
            pointer.remove(); // Delete prevDiff.
            pointer.next(); // Walk past thisDiff.
            thisDiff = pointer.next(); // Walk past nextDiff.
            nextDiff = pointer.hasNext() ? pointer.next() : null;
            changes = true;
        } else if (thisDiff.text.startsWith(nextDiff.text)) {
            // Shift the edit over the next equality.
            prevDiff.text += nextDiff.text;
            thisDiff.text = thisDiff.text.substring(nextDiff.text.length())
                + nextDiff.text;
            pointer.remove(); // Delete nextDiff.
            nextDiff = pointer.hasNext() ? pointer.next() : null;
            changes = true;
        }
    }
    prevDiff = thisDiff;
    thisDiff = nextDiff;
    nextDiff = pointer.hasNext() ? pointer.next() : null;
}
// If shifts were made, the diff needs reordering and another shift sweep.
if (changes) {
    diff_cleanupMerge(diffs);
}
}

```