Software Requirements Specification

for

Simulizer

Prepared By

Matthew Broadway**,** Kelsey McKenna,
Michael Oultram, Charlie Street, Theo Styles

Revision History

Name	Date	Reason For Changes	Version
Matthew Broadway Kelsey McKenna Michael Oultram Charlie Street Theo Styles	29/01/2016	Initial Version	0.5
Charlie Street	14/03/16	Revision near end of project	1.0

Table of Contents

Revision History
Table of Contents
1. Introduction
1.1 Purpose
1.2 Document Conventions
1.3 Intended Audience and Reading Suggestions
1.4 Product Scope
1.5 References
2. Overall Description
2.1 Product Perspective
2.2 Product Functions
2.3 User Classes and Characteristics
2.4 Operating Environment
2.5 Design and Implementation Constraints
2.6 User Documentation
2.7 Assumptions and Dependencies
3. External Interface Requirements
3.1 User Interfaces
3.2 Hardware Interfaces
3.3 Software Interfaces
4. System Features
4.1 High-Level Visualisation
4.1.1 Description and Priority
4.1.2 Stimulus/Response Sequences
4.1.3 Functional Requirements
4.2 CPU Visualisation & Simulation
4.2.1 Description and Priority
4.2.2 Stimulus/Response Sequences
4.2.3 Functional Requirements
4.3 Code Editor
4.3.1 Description and Priority
4.3.2 Stimulus/Response Sequences
4.3.3 Functional Requirements
4.4 Interface
4.4.1 Description and Priority
4.4.2 Stimulus/Response Sequences
4.4.3 Functional Requirements
5. Other Non-functional Requirements
5.1 Performance Requirements
5.2 Safety Requirements
5.3 Security Requirements
5.4 Software Quality Attributes
5.5 Business Rules
6. Other Requirements
7. Risk Analysis
Appendix A: Glossary

1. Introduction

1.1 Purpose

The purpose of this document is to provide a clear specification for a piece of software designed to simulate and visualise the operation of a CPU with a simplified architecture. This document will define the scope of the project and constraints which we are limited to; as well as the target audience for this software and the system's functional/non-functional requirements. At the end of the document, you can find a risk analysis for the project as well as the project's intended development schedule.

1.3 Intended Audience and Reading Suggestions

This document is intended for the use of the development team itself (the 5 authors), but also as a reference for the team's tutor and two course instructors. This document is not intended for the majority of our intended user base; the exception to this may be the module leader for whom we are writing this software for as they may want to check the suitability of the software for teaching purposes. This document can be read in the order in which it is written. For more information on the background on what we are developing as well as our user base, please refer to the following:

Computer Systems and Architecture Module Information: http://www.cs.bham.ac.uk/internal/modules/2015/19340/
Visualising the MMIX Superscalar pipeline: https://www.ncsu.edu/wcae/ISCA2004/submissions/boettcher.pdf

1.4 Product Scope

This piece of software is being written to act as a teaching resource for the Computer Systems & Architecture module taught at the University of Birmingham. The objective is to help students learn the relationship between high level algorithmic thinking and the basic operations of a CPU by simulating the changes in the processor state and visualising the steps of an algorithm being run on it. This will re-iterate algorithmic as well as teaching the basics of computer architecture implicitly through use of the program. The ultimate goal of this project is for students to use the software to aid with assignments, possibly being presented as an optional tool for the module in the future.

1.5 References

Existing Systems:
Spim Simulator: http://spimsimulator.sourceforge.net/
Visualising the MMIX Superscalar pipeline: https://www.ncsu.edu/wcae/ISCA2004/submissions/boettcher.pdf

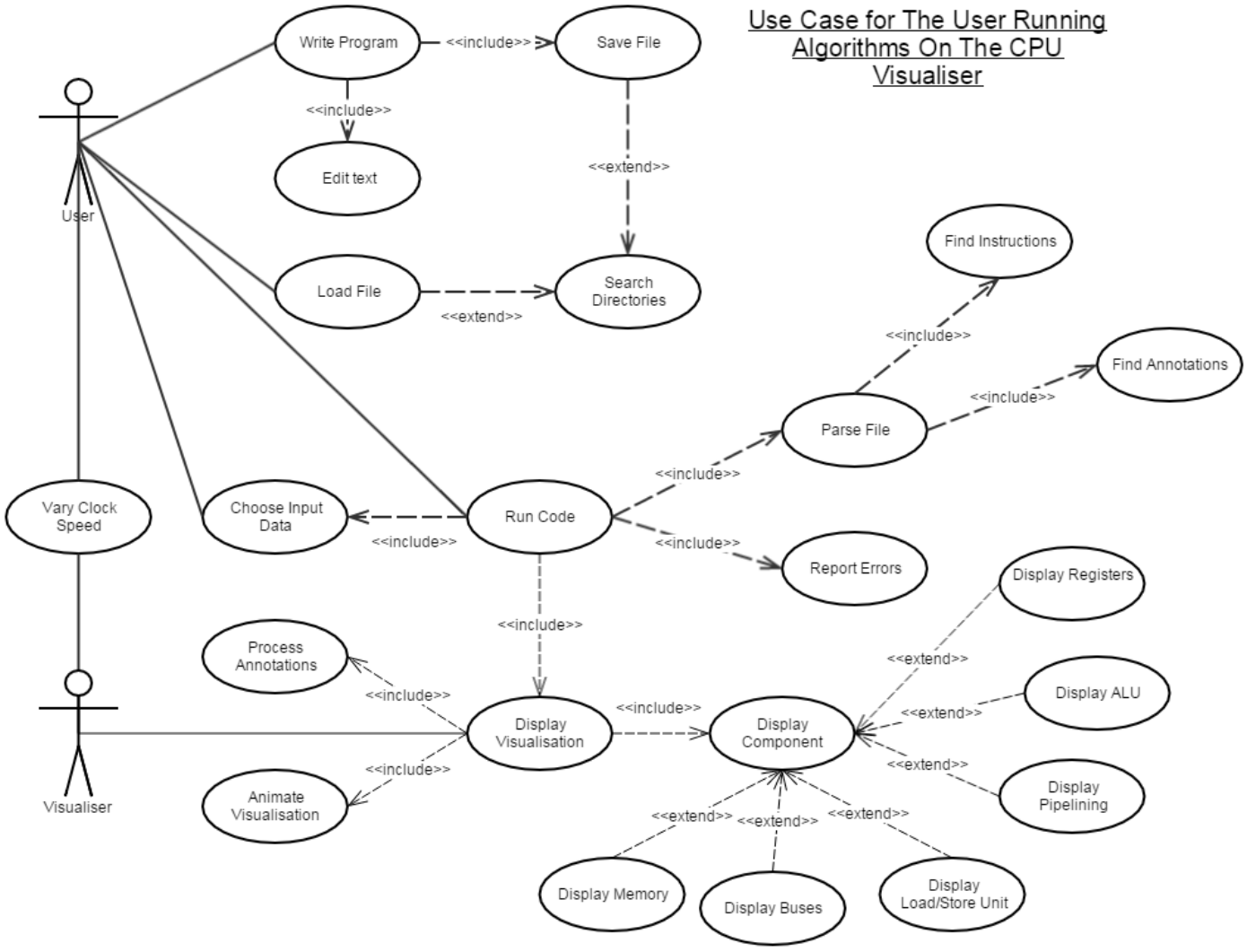
Build Tools/External Libraries
Ace Editor: https://ace.c9.io/
ANTLR: http://www.antlr.org/
Gradle: http://gradle.org/
GSon: https://github.com/google/gson
JFxtras: http://jfxtras.org/
JUnit: http://junit.org/

Information on our User Base:
Computer Systems and Architecture Module Information:http://www.cs.bham.ac.uk/internal/modules/2015/19340/

2. Overall Description

2.1 Product Perspective

The need for this software comes from the missing features in existing simulators. A popular CPU simulator currently used by our target audience in academia is the Spim Simulator (see references 1.5) which simulates a MIPS 4000 processor. Our issue with this is that the only way to monitor the internal state of the simulator is through tables of numbers (values of registers and memory locations). Although this may be effective for professionals who are already familiar with the internal workings of a CPU, we don't believe it is as efficient for new users trying to learn from the simulator. Our simulator aims to provide accurate and clear visuals to convey knowledge that may be applied in other, more complex simulators or real world low-level programming.



(Figure 1: Use case diagram)

2.2 Product Functions

- Provide an editor for a custom simplified assembly language (with a set of JavaScript-based annotations to control the high-level algorithm visualisation). The editor will also provide line highlighting on execution.
- Provide a simulation and visualisation of an abstract simplified CPU, as well as visualisations displaying the operations of the execution pipeline.
- Provide methods of displaying data, as found in other simulators.
- Provide controls for interacting with the simulation during execution such as changing the clock speed, pausing, allowing the user to write code, etc.
- Provide extra information upon interaction with the visualisation. For example tooltips which explain the functions of each component will appear when the user selects the component.
- Provide a high-level visualisation of annotated algorithms currently being run. ie at the level of data structures and structured loops such as for and while, dependent on the detail of the annotated source code.

2.3 User Classes and Characteristics

Students of the Computer Systems & Architecture module: These will be the most frequent users of this system. We expect the software to see semi-frequent use as part of understanding the module content. They would be in a position to use the whole system, with no components requiring access restriction. We can assume the students have the required level of technical skill and knowledge of the high-level algorithms which are built in examples (since this is a 2nd year module). We assume that they have limited hardware knowledge but have at least some basic knowledge of assembly language (even if they don't, this will be fully documented in the system as well as user documentation bundled with the software).

Module lecturer: The lecturer would be expected to use the software significantly less, but we expect would be significantly more adept at its operation. A typical expected use case would be to use the visualisation in place of certain static slides during lectures (but not as a complete replacement). We expect they would also create new algorithms complete with annotations, whereas students may not reach the required level of expertise or familiarity with the annotation language. As with a student, a lecturer would be in a position to use the whole system. The lecturer will have a significantly higher knowledge of the hardware and will have good knowledge of assembly so nothing needs to be catered for them specifically that students could not also take advantage of. They are also assumed to have good technical skills. The lecturer would probably be much more likely to write using the editor for demonstration purposes due to the advantage during teaching situations.

2.4 Operating Environment

The intended operating environment for this software is locally on any systems which can run Java 8 or higher (no network connectivity is required). The product will be cross-platform (usable on Windows, OS X and Linux etc.). The suitability of an operating system is purely dependent on its compatibility with Java 8 and JavaFX8 (other dependencies are bundled with the software). The system used must have a minimum of 1GB free permanent storage plus more for user-created assembly language files (other PC requirements are not known at this point). As a result, this software will not be usable/suitable for more mobile devices (it would be usable on a touch screen laptop; this may not be the easiest (or desired) thing to do given the potential complexity of this software). Upon release of the system, the software will be bundled into a zip file, which contains the .jar file to run the system. Therefore, the user must be able to run jars on their computer to run the software.

2.5 Design and Implementation Constraints

The main constraint of this project is time and resources of the team which will be dealt with internally. There aren't any regulatory policies to follow except for the licenses for some of the external (public) libraries we will be using and the hardware requirements are modest so long as a suitable operating system is used. The software will be usable in English and there are little security considerations beyond the security of the host machine, i.e. the use of code within the simulator will not expose any risk not already posed by it residing in memory. The exception to this is the JavaScript based annotation system, which does use an interpreter. However, there are settings in this to remove the ability to access local files/connect to a web server etc. and these have been switched on to prevent potentially malicious code being run on the system. Due to the nature of this project, maintenance is unlikely to be provided, however high programming standards will be kept at all times to allow updates in the future if necessary.

2.6 User Documentation

Due to the intended usage as a teaching resource, we will provide an extensive user manual for the software. If the software is to be fully deployed and distributed, we will bundle the manual in PDF format with the software. We can also make the manual available online, alongside other module resources. The manual will provide comprehensive instructions/information on:

- Running the simulator and interacting with the visualizer.
- Writing in our simple assembly language.
- How to use annotations within the user-written assembly code to show an abstracted view of what the code is doing in the simulation.

Help will be provided by the software itself if it encounters (detectable and diagnosable) problems with the user's code. Due to the time scale of this project, these messages will likely be much less comprehensive than the bespoke documentation. The user always has the option to translate their code (or hopefully move verbatim) to another simulator in order to receive more comprehensive error messages. However, due to the lack of error messages given in Spim Simulator, Simulizer will aim to provide higher levels of error detail than that.

2.7 Assumptions and Dependencies

The requirements of this project are, at this point in time, dependent on many assumed factors:

- We assume that the complexity of the simulation is obtainable within the given time-frame.
- We also assume that the annotation language produces reasonable visualisations with proportional effort to the algorithm write.

As far as 3rd party software is concerned:

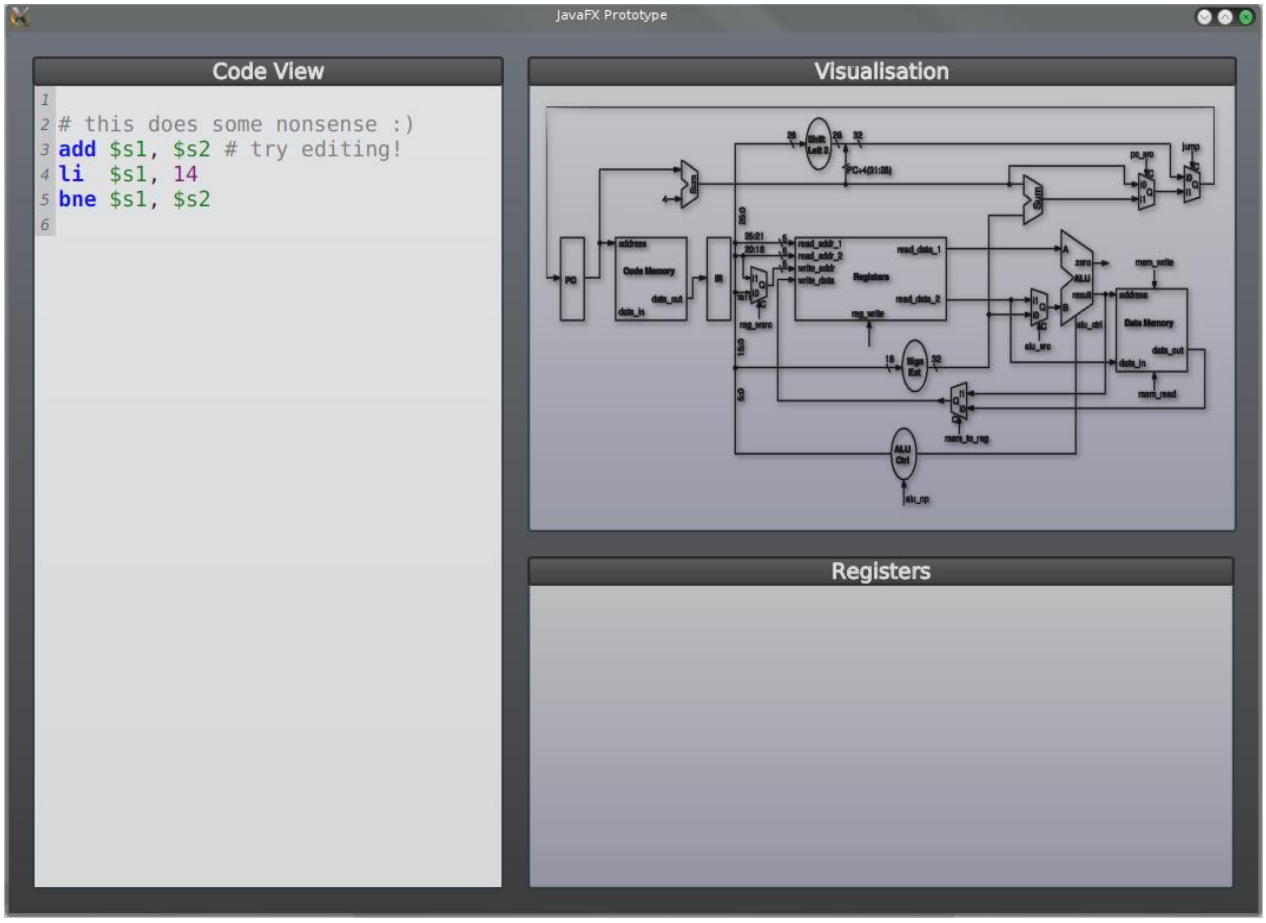
- We are assuming that JavaFX is capable of the level of 2D graphical rendering that is required by the software.
- We also assume that ANTLR is capable of parsing our version of the MIPS language.
- We assume that the Ace editor is capable of being used for the functionality we intend to use it for (line and syntax highlighting).

3. External Interface Requirements

3.1 User Interfaces

The user interface should be extremely reconfigurable and uncluttered to allow for different use cases and clearly convey the information to the user. To achieve this the application will utilise technologies such as JavaFX which allows us to create internal windows within our main application window. These sub-windows will be able to be:

moved, deleted, resized and moved between monitors in a multiple monitor setup. The components of the GUI will be supplied with documentation accessible from the program. Keyboard shortcuts for the main actions will be assigned and documented where appropriate. These settings will be reconfigurable by the user (possibly externally from the application using a configuration file). Error dialogs will be displayed when the simulation itself encounters a problem. If the process being simulated encounters an error then the simulation will not crash, but will instead log the problem to the standard output of the simulated machine. The interface is fully themeable through CSS using JavaFX and the simulator will expose the capability of loading custom CSS files. The application will provide an interface for editing the algorithm to be run on the simulated processor.



(Figure 2: An early prototype using JavaFX)

3.2 Hardware Interfaces

The only peripherals required for the software are a standard mouse, keyboard and a visual display. The application will utilise a multi-monitor setup however this is not required.

3.3 Software Interfaces

JavaFX is a standard library for the Java8 platform which we will be using to build the user interface for the application, allowing the program to run across a wide variety of devices. This library will be used to render components such as the text editor, CPU visualisation, form controls, menus, etc.

We are going to use ANTLR (see references 1.5) in order to generate a parser for our reduced assembly language. When using ANTLR, a grammar is specified in a separate file and then ANTLR generates the Java code required to parse the language. As described in section 2.6, this will allow the user to receive some feedback on errors in their code. It will also enable us to extract register names, values, etc.

Gradle (see references 1.5) is a build automation tool which we will be using to handle dependencies, to help with development across multiple IDEs and operating systems, and to generate build and testing reports. This will be used during development, but will only need to be used for building purposes and will not be needed at runtime. The tool also allows us to provide a source-based distribution of the software alongside the binary distribution.

The ace editor is a publically licensed text editor, which we will be using to provide a usable and comfortable editing environment for the user. This needs to be downloaded onto the user's system; it will be bundled in the downloadable zip file upon full release.

4. System Features

The following is the full set of requirements we intend to implement for this system. All of the requirements are prioritised individually using high/medium/low values to judge their priority. By this priority we are referring to the necessity of this requirement to be implemented in order to make the system be a success. Additionally, for each whole system feature further priorities are specified for the benefit (how useful/necessary it is for the system) and penalty (how severe it would be if the feature wasn't fully implemented). These two values are prioritised on the same scale. By prioritising our system features and functional requirements, it allows us to make more informed decisions when scheduling our development time throughout the project.

4.1 High-Level Visualisation

4.1.1 Description and Priority

Using annotations in the assembly code, data structures, such as lists, trees, graphs, and other data structures will be shown along with the low-level visualisation of the CPU.
Benefit: **High**, Penalty: **Low/Medium**.

4.1.2 Stimulus/Response Sequences

- User annotates assembly code and then chooses to start the visualisation.
- When an annotation is encountered, the appropriate visualisation of the data structure will be shown, i.e. it will look at particular registers and "memory" locations.

4.1.3 Functional Requirements

4.1.3.a: The system will read annotations from the source file, giving it a precise description of the points of interest in memory/registers. **(High)**

4.1.3.b: If an annotation contains anything unknown to the system, such as variables and unknown commands, the system will warn the user. **(Low)**

4.1.3.c: The system visually represents an appropriate data structure for a given algorithm, the data in which is based off of the CPU state. **(High)**

4.1.3.d: The data structure displayed by the visualizer will be animated in accordance with the annotations found and 'sent' to it via a form of message passing. **(Medium)**

4.1.3.e: The visualisation will stay synchronised with the clock but not necessarily strictly in one clock cycle, i.e. swapping two numbers might take three clock cycles. **(High)**

4.1.3.f: The user will be able to choose whether they want the high level visualisations open at any given time. **(Low)**

4.2 CPU Visualisation & Simulation

4.2.1 Description and Priority

The system will accurately display the internal state of the simulated processor at each stage of execution, including sub-tick operations, such as the various stages in the pipeline. Furthermore it will display the interconnections between the processor's components, and where appropriate show the data travelling between them.
Benefit: **High**, Penalty: **High**

4.2.2 Stimulus/Response Sequences

When the user executes a program on the simulator and has the visualisation windows open, the visualisation is produced automatically with no input required from the user, however the user can influence the visualisation through the standard controls, e.g. varying the clock speed. The user can also play and pause the visualisation as well as step forward tick by tick. Additional interactions can be made via the use of provided tooltips.

4.2.3 Functional Requirements

4.2.3.a: The visualisation will display and simulate the following components of our abstract CPU:

- Program counter
- Arithmetic and Logic Unit
- Instruction register
- General purpose registers
- Simplified buses
- Simplified memory

4.2.3.b: The components of the CPU, when selected, will display a description of the purpose/use of that particular component. **(Low)**

4.2.3.c: The visualisation will animate the transportation of data/instructions throughout the processor during execution. This will be primarily through the activity of buses. **(High)**

4.2.3.d: The visualisation will contain abstractions to provide a clearer description of the features we deemed more important, e.g. a collection of related buses will be shown as a single connection between components. **(High)**

4.2.3.e: The visualisation, including animations, will be synchronised with the internal clock of the simulation. The animations will be adaptable depending on what is feasible given the clock speed. **(Medium)**

4.2.3.f: The visualisation can be played, paused, and the user will have the option to step forward tick by tick. **(Low)**

4.2.3.g: The program will be run using a pipeline superscalar architecture, executing a MIPS-compatible RISC instruction set with significantly fewer instructions. The system will not use speculative execution. **(High)**

4.2.3.h: The execution of a pipelined CPU will also be visualised, displaying information such as the instructions in each stage of the pipeline as well as information about pipeline 'hazards'. **(Medium)**

4.3 Code Editor

4.3.1 Description and Priority

The system will provide a code editor, which the user can use to open, edit, and save files as well as add annotations.
Benefit: **Medium**, Penalty: **Low**

4.3.2 Stimulus/Response Sequences

While the user is running the program, they will click on a button, opening the text editor in a new window/frame. User can then close the window while the visualisation is running.

4.3.3 Functional Requirements

4.3.3.a: The user will only be able to open and save files with the ".s" extension - the extension for MIPS programs. **(Medium)**

4.3.3.b: The user will be able to choose between their existing programs they have written for the software (or create new ones), or open demo files bundled with the system. **(Medium)**

4.3.3.c: The user will be able to save their own programs as well as make updates to the bundled ones. **(Medium)**

4.3.3.d: Syntax highlighting will be present on the assembly code, making code editing a simpler process. **(Low)**

4.3.3.e: Line numbers will be shown to help with locating errors identified by the logger. **(Low)**

4.3.3.f: The code editor will include a primitive logger/error checker, indicating that there is an error at a particular line number. **(Low)**

4.3.3.g: There will be a run button in the menu bar, which will cause the code to be parsed and then, if successful, will start running on the visualizer. Any existing, running simulation must be halted (manually or by finishing execution) before a new one is run. **(High)**

4.3.3.h: If the user tries to close the code editor having not saved, the system will prompt asking if they wish to save or not. **(Low)**

4.4 Interface

4.4.1 Description and Priority

The user will be presented with a main window which brings together sub-windows displaying content for the various components.
Benefit: **High**, Penalty: **High**

4.4.2 Stimulus/Response Sequences

The interface will be shown on start-up with a default set of widgets, including the code editor, the CPU visualizer, etc. Once the user is finished with the system, closing the interface will stop running processes.

4.4.3 Functional Requirements

4.4.3.a: Provides windows for the major components previously mentioned (code editor, CPU visualizer, data structure visualisation). **(High)**

4.4.3.b: All components in the main window will be reconfigurable with respect to size and positioning. New windows can be added or removed at the user's wish. **(Medium)**

4.4.3.c: The interface will allow changing of colour schemes by specification of the user, based upon a set of files bundled with the software. **(Low)**

4.4.3.d: If the user attempts to close the application while the code editor is in use, they will be presented with a prompt confirming whether or not they wish to leave. **(Low)**

4.4.3.e: The main window will have a menu bar, allowing the user to carry out tasks, such as change colour scheme, open new windows, open files, exit the system, run code, etc. **(High)**

4.4.3.f: The user will be able to save the current layout of the widgets to an external configuration file, which can then be selected when the user returns to the system at a later date. **(Low)**

4.4.3.g: A full options menu will be provided with the system, to allow the user to change their preferences of the system in one place. **(Low)**

5. Other Non-functional Requirements

5.1 Performance Requirements

PR-1: The user interface will be fully responsive (i.e. it will be thread safe) such that the user is always able to use the system without any significant freeze/lag time.

PR-2: The underlying software will be fully consistent and responsive during its execution (i.e. consistent clock speed).

PR-3: During normal use, the clock determining the speed of execution will be accurate to within a 5ms error range. This will keep the system running as intended as any large speed variation would break the synchronisation of the system.

PR-4: The system will visualise/run the user's self-written code accurately i.e. the program will successfully execute what they 'intended' to write. This has a very high priority for completion.

5.2 Safety Requirements

Due to the absence of any networking capabilities and sensitive data, there are very few safety issues with this product. Furthermore, the fact that the CPU is only being *simulated* means that the internal state of the host machine cannot be maliciously altered.

SR-1: The system will be fully isolated from the simulated CPU. That is, the CPU we are simulating will not have any effect on the internal state of the host machine's CPU except for those that would occur as a result of the standard execution of a safe java program. By this it is meant that the tasks being carried out on the simulation will not be directly carried out on the host's CPU, it will instead be run through Java.

SR-2: Usage of the JavaScript annotations will be restricted, preventing file access, etc. This will prevent certain types of malicious code been executed on the host machine.

5.3 Security Requirements

For the same reasons as in Safety Requirements, there are very few security issues regarding this system.

SEC-1: Much like in the Safety Requirements, the JavaScript annotations/interpreter, will be restricted in what can be run. This prevents potential security issues such as an XSS attack within a malicious program, as nothing harmful can be executed within this system.

5.4 Software Quality Attributes

Correctness-1: All educational explanations or information given by the system to the user will be clear and accurate to the abstract level at which we are working. This attribute has a high preference due to its link to the entire purpose of the system.

Usability-1: The GUI given to the user will be pleasant to look at, with well thought out colour schemes selected to maximise the user's experience with the system (such as colour schemes to account for dyslexic or colour blind users for example). The interface along with the educational content has our highest preference for the non-functional requirements.

Usability-2: The system will be very simple/easy to use, with comprehensive user documentation given when some part of the system needs to be explained. The preference of this requirement goes hand in hand with *Usability-1*.

Usability-3: The user documentation for the system will be easily accessible to them at any point during the use of the program, whether this is by it being embedded in the software or given as a link in the program. The other attributes have higher preference as this only concerns where we store the documentation.

Portability-1: The system will be portable across different Operating Systems; we will be considering popular operating systems such as Windows, Mac OS and Linux specifically, though alternative OSs may be compatible.

Robustness-1: All potential errors that occur due to the system (most likely due to incorrectly written user code) will be caught and dealt with sensibly. These errors should not significantly detract from the user's experience of the software. The preference for such a requirement is high, however the way in which it is 'dealt' with may vary.

Flexibility-1: All of the GUI components available to the user follow some common theme where the theme is changeable by the user. This has a low preference since it is really considered a 'stretch' requirement for the project.

Maintainability-1: The large majority of this software will be written with Java 8 and JavaFX 8 for user interface (as an alternative to Swing). The JFxtras library will also be used for extra interface tools. These are up to date versions of these and so it will be maintainable for a relatively long time.

5.5 Business Rules

Any user of this system has full access to the system, including the ability to change our example code for the visualisation. Therefore there are no restrictions on what roles can be performed by any person using the system.

6. Other Requirements

Legal-1: Due to us accessing the user's filesystem, we will keep in accordance with the Computer Misuse Act (1990) such that we don't access any data which we don't have explicit permission to use (we will create our own folders for code etc.), regardless of the access permissions in place on the file(s). We will also ensure no unauthorised modifications occur on the system.

7. Risk Analysis

In order to ensure this project can be successfully produced and implemented within our resources budget (5 team members) and time constraint of the week commencing 21/03/2016, we will use team communication tools, such as Slack, a project schedule, and keep track of our tasks using Taiga for user stories. In order to achieve our goals for the project we need to comprehensively cover everything that could possibly go wrong during the production cycle. Due to the likelihood of at least one of the issues covered in this analysis occurring being incredibly high, we will need to develop comprehensive methods of preventing/avoiding errors and in some cases providing contingency plans so as to reduce time wasted due to an unexpected issue arising.

These risks can fall into multiple categories and all will be covered. These consist of risks due to people, the software/hardware/services we are using, the development cycle itself, scheduling, user response and also in our case, risks associated with the concepts we are planning to implement.

As well as some method of approach to each of the risks identified, each shall be labelled with a measure based upon the likelihood of it happening and the impact this would have on the project being a success. The impact will be weighted higher and the higher the value associated with the risk, the more important it is to consider. The risk value is calculated by: $(1.5*\text{impact})^2 + \text{likelihood}^2$ where both are measured from 0 to 10.

Risk:	Being late for schedules and generally running out of time to fully finish the project as we intended.				
Category:	Scheduling/ Development	**Risk Value:**	208	**Type of Approach:**	Avoidance, compromise & project management.
Strategy:	This is possibly the biggest risk of the entire project; we are working under a very strict schedule. For this reason, there will be multiple layers for dealing for this risk. The first is appointed to good scheduling of the project. We will be following an agile method of development allowing us to schedule in small, quick parts. As well as giving more accurate control over development length, it also allows us to develop the software in increments. This means even if we are running short on time we should still have a working piece of software. The software itself will be modularised so the				

	absence of a component will have minimal effect on the running software. As a last resort, we look at the remaining functionality, which is all prioritised, and we cut out and remove low priority features to fit with our schedule.

Risk:	We become absent of a team member for a prolonged length of time.				
Category:	People	**Risk Value:**	145	**Type of Approach:**	Compromise, rescheduling.
Strategy:	This is probably one of the most unlikely of the risks being discussed. However, it would be crippling to the team if we were to decrease in number for such a large project. In this scenario, we would have to remove a large amount of functionality (what amount depends on how far we are into the project) and go from there. If this did actually occur, we would also have to fully discuss our future plans with tutors/course instructors.				

Risk:	Struggling to implement certain features of the software due to theoretical knowledge				
Category:	Understanding of Concept	**Risk Value:**	105.25	**Type of Approach:**	Discussion of content with knowledgeable people and research.
Strategy:	Our user base for this software is based around the ‘Computer Systems & Architecture’ module, a module we are studying at this current moment in time. Therefore, when getting into deep implementation details, there may be gaps in our understanding. All the resources for the module should suffice. Failing this, we have had discussions with Ian Batten, the course leader and he is willing to give us help/advice if we come to such points. This should help us have sufficient understanding to implement the software.				

Risk:	Parts of the system don’t work				
Category:	Development/ Implementation	**Risk Value:**	81.25	**Type of Approach:**	Modularisation/ compromise
Strategy:	When writing the software, we will aim to utilise good software design patterns such as model-view-control. Through this and a thorough, comprehensive design we aim to keep the code highly modular; this means any new functionality will rely very little on the existing software and vice versa. This means that, in the case of having to cut our losses on a piece of (low priority/importance) functionality, we can choose to compromise via simplifying it or removing it entirely.				

Risk:	Conflicts within the group.

Category:	People	**Risk Value:**	69.25	**Type of Approach:**	Compromise, democratic
Strategy:	Within this project, there will be many different sub-areas which need to be worked on. We can assign one person to be in charge of a certain area. This person has a higher weight and say on any decision concerning that area. When conflicts arise, we will deal with them prominently. We will deal via discussion followed by a vote on any compromises. On any vote, the person in charge of the area being voted on has their vote worth 'more'.				

Risk:	Potential users of the software don't like the interface/ find it hard to use.				
Category:	User response /design	**Risk Value:**	29.25	**Type of Approach:**	Discussion with users/careful design and prototyping.
Strategy:	When looking at similar software to that of which we are trying to implement, we have noticed they are all lacking usable, clean interfaces. Therefore, one of our top priorities is to make this software easy to use and understand (from an educational perspective) and also to make it easy on the eye. We will do this by repeatedly prototyping the interface and then, due to the ease of contact between us and the users, gathering feedback such that we create and design a piece of software that people actually *want* to use.				

Risk:	SVN server goes down/losing access to all of our data locally				
Category:	Services	**Risk Value:**	25	**Type of Approach:**	Full avoidance
Strategy:	In order to be fully protected with our source code/documentation we have decided upon using two different version control systems to allow all of our data to be stored remotely in two locations (as well as on all of our respective local machines). We will be using Git alongside SVN. In doing this, one team member will be a gatekeeper to the SVN repository, and so the software will be remotely stored in two places. This completely reduces any chance of us fully losing our software due to some malfunction. Also, we will encourage the regular pushing of work in case of local work losses.				

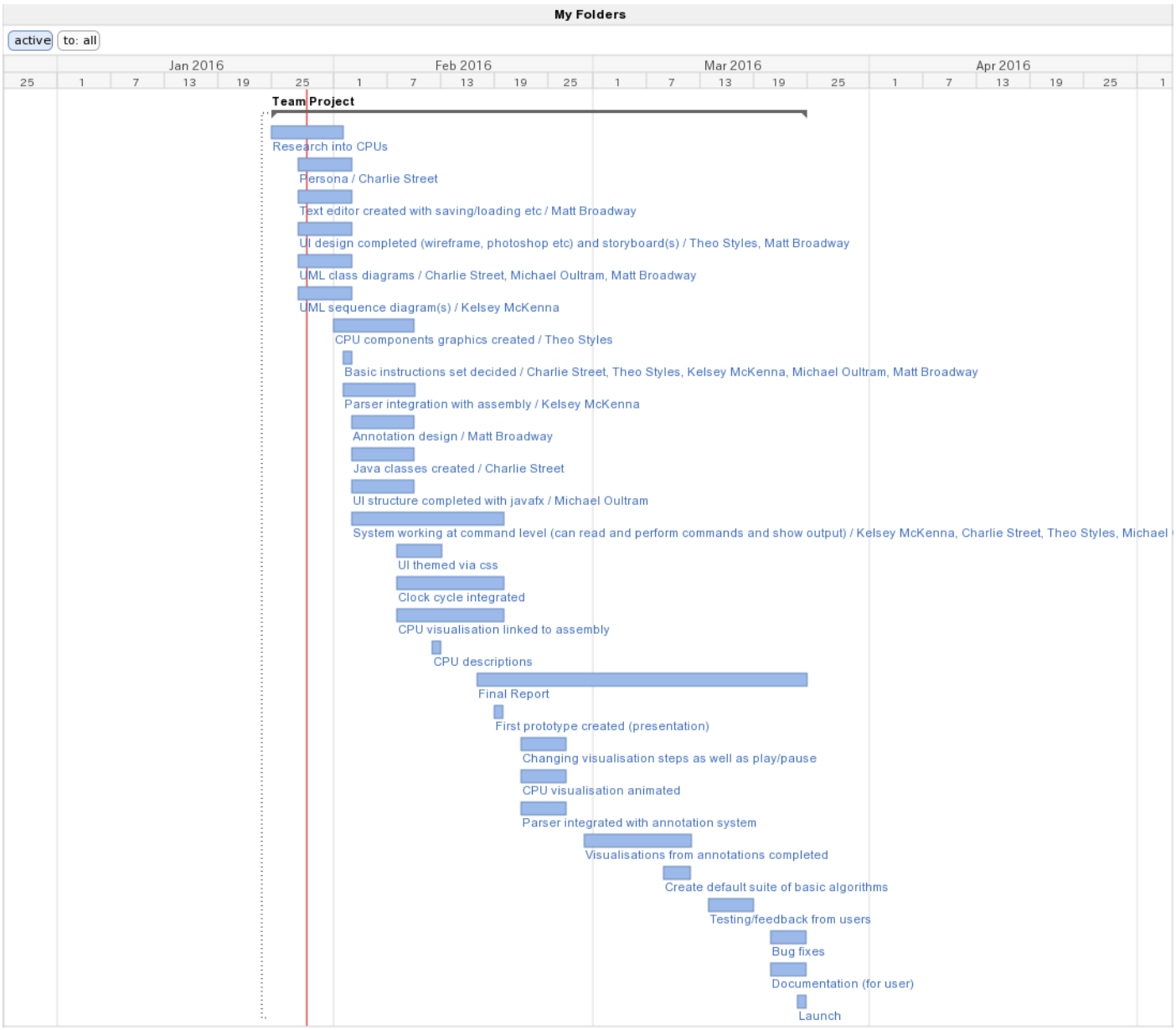
Risk:	Problems linking functionality together				
Category:	Development/ Implementation	**Risk Value:**	25	**Type of Approach:**	Avoidance
Strategy:	For the reasons stated above, we will try and modularise the code as much as possible through the extensive use of good software engineering practices such as design patterns. This will mean that, even with the highly independent sections of the software (written by different team members), there should be no real difficulty in joining in our code. (One example in avoiding this may be through the use of Java interfaces (a skeleton for a class/classes whereby only the method headers are given), one				

	can write the implementation whereas another can 'use' it without even knowing whether it currently exists).

Risk:	Portability issues between Operating Systems				
Category:	Development/ Implementation	**Risk Value:**	18	**Type of Approach:**	Consideration of software used
Strategy:	The very large majority of code for this software will be written in Java, using JavaFX for the user interface. To our knowledge, there is no known compatibility issues for this language and library. There will be a small amount of code written in JavaScript, but again, this is not anything particularly obscure, so as long as we take this into consideration, the risk of this hindering our project is very low. Should we come to the deployment stage, different executables may have to be produced for different operating systems but that wouldn't be a complicated task (particularly considering that, as a team, we have three major ones readily accessible).				

8. Project Schedule

On the next page is a Gantt chart for this project. All events have been organised in terms of their start and finish date. However, we have decided to only assign jobs for the first few weeks. Due to the agile methodology we are using, tasks are relatively short and so in case of situations changing, we can assign new people to tasks nearer the times at our team meetings. The scheduling of our tasks is primarily based upon the priorities we have set for the functional requirements. In some cases, certain lower priority tasks may be completed in-between high priority tasks. This could be due to one of two reasons: the first is that a team member may have spare time between sprints and so it makes sense to carry out a smaller task within that time frame. The second is that a specific requirement may almost be a by-product of a larger requirement. This is likely to be apparent when it comes to developing the text editor; the opening and saving files may almost complete the requirement for editing user created files or the demo files for example. Please note that the 'Testing/Feedback from Users' is more at a level of the final report and final acceptance tests, other forms of testing will be carried out throughout the project, particularly due to the use of an Agile methodology.



Glossary

Definitions

Simulation: the simulation in the system refers to the underlying representation and operation of our abstract CPU.
Visualisation: this is the actual display of the CPU, including animations representing the actions occurring in the underlying representation.
High-level visualisation: this refers to the visualisation of the data structure(s) the CPU is working on, e.g. lists, graphs, etc.
Pipeline: This refers to a method of executing programs in a CPU whereby in one cycle, one instruction is executed, while the next is decoded, and the one after that is fetched. This is a general method for speeding up execution, it does add complexity to the execution however.