Team Project - Team C5

Software Requirements Specification

Version 1.0

29.01.2015

Thomas Clarke, Mihnea Patentasu, Piotr Wilczyński, Danyil Ilchenko, Owen Pemberton, Botond Megyesfalvi

1. Introduction

1.1 Purpose

This purpose of this Software Requirements Specification (SRS) is to provide a detailed overview of our project based on transforming regular expressions into deterministic finite automata. It describes the aim of the project and its requirements. It is to explain the user interface and possible interactions with the user. This document is intended for review by relevant lecturers (as they will be using it for demonstration purposes) and for the production team as a basis for development.

1.2 Scope

The purpose of our system is to develop a learning tool to help Computer Science students learn about Finite State Automata and understand the Thomson's Construction and Subset Construction algorithm using an interactive visualisation. Users are able to input regular expressions and visualise the steps to transforming them into deterministic finite automata.

1.3 Definitions, Acronyms and Abbreviations

1.3.1 Definitions:

Deterministic Finite Automaton **(DFA)** — a finite state machine that accepts/rejects finite strings of symbols and only produces a unique computation (or run) of the automaton for each input string. 'Deterministic' refers to the uniqueness of the computation.

Nondeterministic Finite Automaton **(NFA)** — unlike a DFA, it is non-deterministic, i.e., for some state and input symbol, the next state may be one of two or more possible states.

1.3.2 Acronyms:

DFA: Deterministic Finite Automaton

NFA: Nondeterministic Finite Automaton

GUI : Graphical User Interface

1.4 References

IEEE. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

Ken Thompson (Jun 1968). "Programming Techniques: Regular expression search algorithm". *Communications of the ACM* **11** (6): 419–422.

Rabin M. O.; Scott D., (1959). "Finite automata and their decision problems". *IBM Journal of Research and Development* **3** (2): 114–125.

Xing, Guangming. “Minimized Thompson NFA”, [available at: http://people.wku.edu/guangming.xing/thompsonnfa.pdf]

1.5 Overview

The rest of this document shall detail a general description of the overall product that shall be produced, along with more specific requirements about the system. This shall also cover how the user will interact with the system. Furthermore, a risk analysis shall be supplied to help solve issues as they arrive as various protocols shall already have been discussed and decided.

The SRS has been organised in the sequence listed above, so that it follows a logical order or describing the project, then explaining various parts of it and finishing with information to be used when actually building it.

2. General Description

2.1 Product Perspective

The system will be a standalone learning tool to be used by university students and lecturers in modules closely related to automata theory. The system will be in the form of an application runnable on Linux, Mac and Windows desktops and laptops. The application will visualise the process of constructing automata from regular expressions, determinising nondeterministic automata, minimising automata and determining automata to be bisimilar.

2.2 Product Functions

With this application the user will be able to build a finite state automata. The user will type in a regular expression, the system will confirm whether it is valid or not, and the application will build the automaton. The user will be able to watch this whole process. The application will display the construction, going step by step with a certain speed that can be changed at any time, or even completely stopped by the user. The construction will pause at relevant stages of the construction (after generating the nondeterministic automata, after removing ε moves, after determinising, after minimising). "Next" and "Previous" buttons will also be included, which will allow the user to go through every small step of the construction manually.

At any point during this construction process a saving and a printing option will be available, which will save as an image file or print the current automaton, with useful text added, which describes the current stage of the constructed automaton, and also the regular expression that it represents. The user will have the option to select the destination folder where to save the image, or view the print preview, and edit printing options.

In case of large automatons the application will offer the possibility to zoom in and zoom out, allowing the user to see, and fully understand every part of the automaton.

After the finite state automata is built, the user by introducing a string will be able to check, whether it is part of the language that the regular expression describes (gets accepted by the automata) or not. The whole process will be displayed, the same way as the construction process, and will have the same functionalities.

Given two regular expressions, the application will be able to build the 2 automata, and check whether they are bisimilar, visualizing the process step by step.

2.3 User Characteristics

Intended users of the system are university students and lecturers involved in automata theory modules. Although there are two types of users for the system, they do not form distinct groups based on how they will interact with it and thus share requirements.

Users will be able to run automata building algorithms on regular expressions as well as determinising, minimising algorithms and bisimilarity checks on automata to observe the visualisation of step by step execution of said procedures. Users will also be presented with options to save or print the resulting automata.

2.4 General Constraints

The application is constrained by the complexity of DFA which are built as a result of NFA determinization. The process of determinising NFA has $\Theta(2^n)$ time complexity and can result in DFA consisting of at most 2^n states (where n is the number of states in NFA). Considering that the application is intended to be run on desktops with screens of standard size, visualizing DFA for large n may not be feasible.

Saving feature of the program is limited by the amount of memory available on a computer at any given time.

2.5 Assumptions and Dependencies

It is assumed that the application will run on a system, that has enough memory on the hard drive to save one or multiple images, otherwise the save functionality won't perform as expected.

It is also assumed that the computer, that runs the application, has a keyboard and a mouse, the interface strongly relies on those, and without those the program would become unusable.

For automata to be printed, a working printer connection is required.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interface Overview

The user interface will mainly consist of a single window where all input, interact buttons, and output/visualisation takes place. The user interface should be tabulated, so that the user can have multiple strings open at the same time. The user interface should switch to a ‘split-view’ model when comparing machines for bi-similarity.

The only other windows will be Save/Open/Export as Image dialogs.

The following is a complete list of input/output features that will be on the user interface.

Type	Component Type	Name	Description
Input	Textbox	Regex Input	Allows the user to input a regular expression.
Input	Textbox	Test String	Allows the user to input a test string. The test string is checked for acceptance automatically after every character entry.
Input	List	List of stored regex strings	A list of stored regular expressions for quick access
Visual Output	Graphical	Visualisation	The visualisation window for the finite state machine.
Visual Output	Graphical	Test String result	Displays whether the test string is acceptable.
Visual Output	Text/Graphical	Current Step	Displays/describes the current step or operation currently being performed.
Menu	Menu Bar Item	New	Opens a new tab with a blank document.
Menu	Menu Bar Item	Save	Saves the current/active document.
Menu	Menu Bar Item	Open	Open a document in a new tab from a file.
Menu	Menu Bar Item	Export	Export the current/active document as an image.

Menu	Menu Bar Item	Print	Print the current/active document.
Menu	Menu Bar Item	Switch to Bisimilarity Checking	Allows the user to select two open machines and switches the GUI to a split-view to check two machines for bisimilarity.
Input	Button	Play/Pause	Toggle button for playing or pausing the animation.
Input	Button	Back Step	Take a step back in the animation.
Input	Button	Forward Step	Advance a step forward in the animation.
Input	Button	Reset	Reset the animation back to the start.
Input	Button	Minimise Automata	Skip to the ‘Minimise Automata’.
Input	Button	Remove Epsilon moves	Skip to ‘Remove Epsilon Moves’.
Input	Button	Determinise	Skip to Determinising.
Input	Button	Auto-play	Button to toggle auto-play.
Input	Button	Store Regex	Store the current regular expression in the in the regex list for quicker access later.
Input	Scrollbar	Zoom	Scroll bar to zoom in/out for the visualisation window.
Input	Scrollbar	Animation Speed	Sets the animation speed.
Input	n/a	Drag to Move	The user can drag with the mouse inside the visualisation window to move the FSA around the window.
Input	n/a	Scroll mouse to zoom	The user can zoom in the visualisation window using the mouse’s scroll wheel.

3.1.3 Hardware Interfaces

The system will use a mouse (for interacting with the GUI), a keyboard (for inputting into text boxes, etc.), a screen (for visual output of the GUI and visualisation) and a printer.

3.2 Functional Requirements

3.2.1 Inputting a regular expression

3.2.1.1 Introduction

The user should be able to input any regular expression.

3.2.1.2 Input

Only the following characters should be used:

- Vertical bars (|) which separate alternative elements
- Parentheses which indicate the scope and the precedence of the operators
- Letters or numbers - the user should be able to input either letters only or numbers only, which will constitute the alphabet of the language represented by the regular expression
- Asterisks (*) which indicate zero or more occurrences of the preceding element

3.2.1.3 Error Handling

The user should not be able to input invalid characters i.e. characters not listed above. An appropriate message should be displayed to inform the user that a certain character is not allowed and to provide the user with a list of allowed characters.

3.2.2 Converting a regular expression into an automaton

3.2.2.1 Introduction

The regular expression inputted by the user should be converted into a nondeterministic finite automaton using Thompson’s Construction Algorithm.

3.2.2.2 Processing

The conversion should be displayed step by step so that the user can see exactly how the algorithm works. The user should be able to specify the speed of the animation and to rewind it.

3.2.2.3 Output

After displaying the animation, the resulting nondeterministic finite automaton should be left on the screen.

3.2.3 Removing epsilon moves

3.2.3.1 Introduction

The user should be able to tell the program to remove epsilon moves from the automaton.

3.2.3.2 Processing

An animation should be displayed showing step by step how epsilon moves are being removed.

3.2.3.3 Output

The resulted automaton with no epsilon moves should be left on the screen.

3.2.4 Determinisation

3.2.3.1 Introduction

The user should be able to tell the program to determinise the automaton.

3.2.3.2 Processing

An animation should be displayed showing step by step how the automaton is being determinised.

3.2.3.3 Output

The resulted deterministic automaton should be left on the screen.

3.2.3.4 Error Handling

If there are still epsilon moves in the automaton, an error message should be displayed to inform the user that epsilon moves have to be removed first.

3.2.5 Saving an image of the automaton

3.2.5.1 Introduction

The user should be able to save an image of the automaton onto the hard drive. The user should be able to specify the filename and the location.

3.2.3.3 Output

The automaton should be saved onto the hard drive.

3.2.5 Printing an image of the automaton

3.2.5.1 Description

The user should be able to print an image of the automaton.

3.2.6 Viewing the automata

3.2.6.1 Description

The user should be able to explore the automaton after animation. Buttons for zooming in and out should be provided and if the automaton does not fit in the window, scroll bars should appear. The user should be able to add more tabs in the window to convert other regular expressions into automata without using the first one.

3.2.7 Bisimilarity

3.2.7.1 Introduction

The user should be able to check whether two automata are bisimilar or not.

3.2.7.2 Output

A window should be displayed informing the user whether the two selected automata are bisimilar or not.

3.2.8 Minimisation

3.2.8.1 Introduction

The user should be able to tell the program to minimise the automaton.

3.2.8.2 Processing

An animation should be displayed showing step by step how the automaton is being minimised.

3.2.8.3 Output

The resulted minimal automaton should be left on the screen.

3.2.3.4 Error Handling

If the automaton is not deterministic yet, an error message should be displayed to inform the user that the automaton has to be determinised first.

3.3 Use Cases

3.3.1 Use Case 1

Name	Convert regular expression to automata
Initiator	User
Goal	Input a regular expression and have the system convert it to an automata.

Main Success Scenario

1. User types in regular expression.
2. System interprets regular expression.
3. System draws nondeterministic automata, with ε moves (with animation).
4. System removes ε moves (with animation).
5. System determinises (with animation).
6. User presses ‘minimise automata’ button.
7. System minimises the automata (with animation).

Extension

1. User presses the ‘load’ button.
    1. System fetches and presents all currently saved regular expressions.
    2. User clicks on a regular expression they want.
    3. System loads the chosen regular expression into the input box.
2. Regular expression is invalid.
    1. Ask user to re-enter a valid regular expression.
    2. Resume 1.
4. The user unticked the ‘auto-play’ button on the start screen.
    1. System stops performing actions and waits for user input.
7. Automata is already in minimal state.
    1. Show decision process anyway, as it is important to show why it is already minimal.

3.3.2 Use Case 2

Name	Compare two automata for bisimilarity.
Initiator	User
Goal	From two regular expressions, check for bisimilarity.

Main Success Scenario

1. User specifies two regular expressions (the second can be given after the first has been interpreted).

2. Build both automata (see use case 1).
3. System shows how it checks for bisimilarity on both automata.
4. System outputs if they are bisimilar or not.

3.3.3 Use Case 3

Name	Save the current regular expression for quick access later.
Initiator	User
Goal	Save the regular expression for quick access later.

Main Success Scenario

1. User presses the ‘save’ button.
2. The system presents the save slots to the user so they can choose where to save it.
3. The user selects a slot.
4. The system saves the regular expression string in the specified slot.

Extension

3. The slot already contains a saved regular expression.
    1. Alert the user that the slot is already in use.
    2. Allow the user to choose whether they wish to overwrite it or not.
4. The system attempts to save the data but no save file is present.
    1. The system creates a file.
    2. The system writes the data to file.

3.4 Non-Functional Requirements

3.4.1 Performance

The system should present a smooth experience. From what we have seen online, efficient algorithms can be written at least for doing generating the initial automata, removing ε steps and determinising. Therefore, the only part of the system that will take up much of the user's time will be the animations. All background processing should be done extremely quickly so the user can be taken straight into the visualization.

It is worth considering that to achieve processing the algorithms without disrupting user experience, they are done in separate threads, allowing the system to pre-generate all of the moves the animation shall do. Keeping a history of this is

important too, as we want the user to be able to speed up, pause and slow down the visualization dynamically at any time. Therefore, having an efficient system is critical.

Furthermore, memory usage isn't a large factor. Most computers have large memory resources and (with more help from Java's garbage collector) using lots of memory shouldn't compromise performance, which is useful as part of achieving the efficiency described above will probably involve storing a lot of actions in memory.

The graphics should be smooth and the interface constantly responsive not only based upon efficient computations but also in terms of the power of the system it is running on. The drawing should be easily rendered so less powerful presentation computers or devices will have no issue showing it so it is accessible to everyone.

3.4.2 Reliability

The system should remain consistent and produce accurate results all of the time. It should be consistent enough so that if a user submits the same regular expression multiple times, the exact same automata, from an identical build process, should be formed. The program should not crash and should scale to fit any window size.

3.4.3 Availability

The system will be completely offline (there won't be a licenced login or any DRM checks) meaning if the user has the program (and an update-to-date version of Java) they should be able to use it without any issues. Given it is reliable, the user will always be able to use it.

3.4.4 Maintainability

The program should be built well to allow easy implementation of new features. This may be achieved through the heavy use of interfaces. To support this, a suitable way to test the program should be fully implemented so that any new features can be validated and then committed to the complete build of the system.

Everything from design to final implementation should be well documented (including Java-Doc) so that whoever works on the system should be able to easily understand what all of the components do and where or how they should implement their new feature.

3.4.5 Portability

The system is to be written in Java, so should in general be compatible with any platform that supports Java. The main targeted systems are Windows, Mac and Linux which should be easy to make the system run on.

4. Change Management Process

The SRS will be updated constantly. Changes can be submitted by any member of the group and they will be visible immediately to all members. Changes will be discussed and approved or rejected at group meetings by means of consensus.

5. Risk Analysis

Scales will be used here and work as follows:

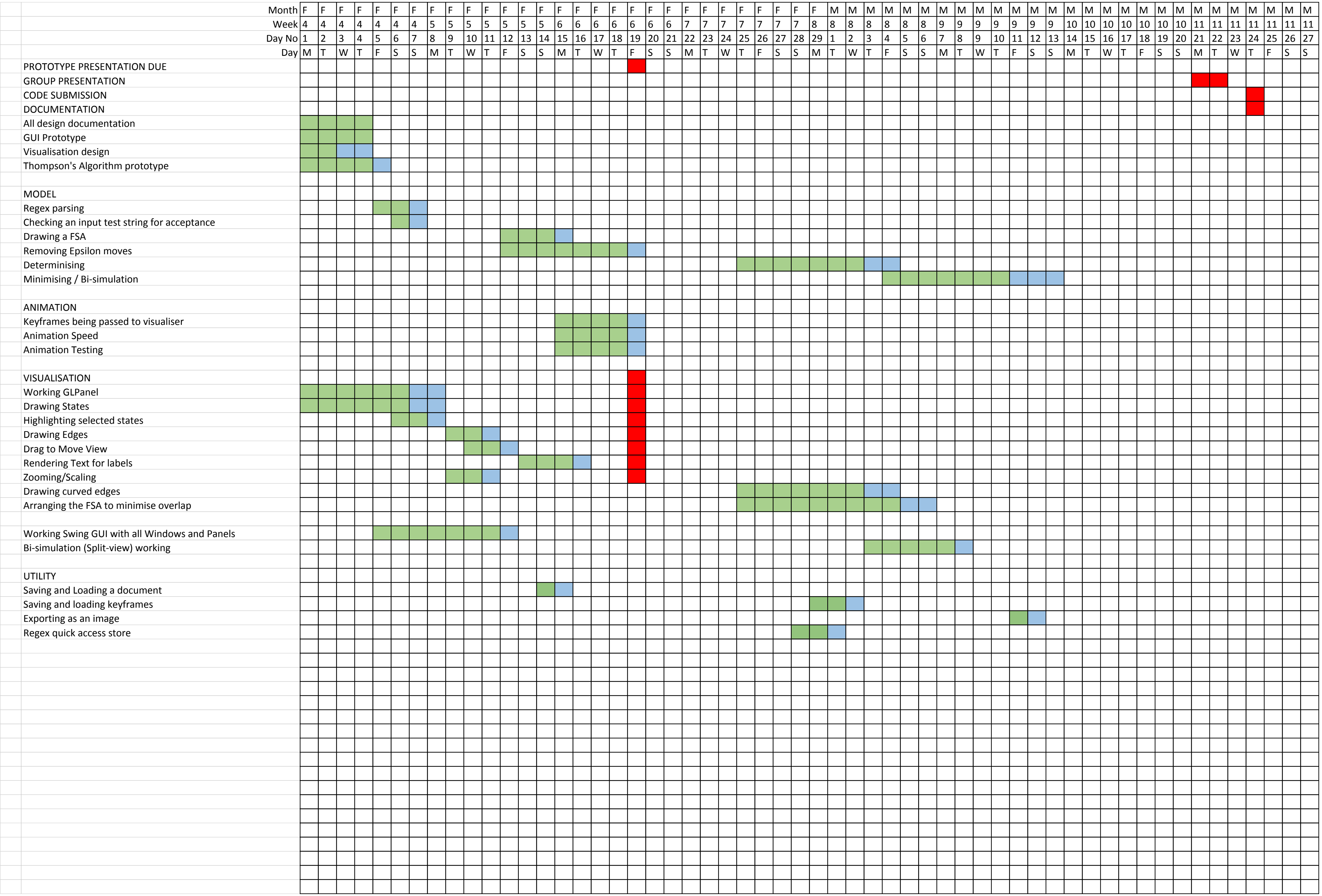
- Probability: 1 (low likeliness) to 5 (very likely)
- Effect: 1 (minimal effect/disruption) to 5 (large effect/disruption)

Risk	Probability	Effect	Strategy Type	Strategy
Computer malfunctions	2	1	Avoidance plan	Ensure each team member can quickly switch from using their personal machine to a lab machine and vica-versa so if one system fails the team member can continue to work.
SVN connection issues	5	3	Contingency plan	If a team member (or all) cannot access the SVN repository, the team shall share code via other media and continue to work until it is restored and the team can all re-synchronise together.
Illness / unavailability to work	3	4	Minimisation strategy	For each piece of code, at least 2 people should fully understand it, so work can continue. Also, ensure everything is committed to SVN so everyone can access it.
Compatibility issues on various operating systems	1	3	Avoidance plan	Use Java which is can be used on almost every system, and use a few external libraries as possible to minimise risk of incompatibility. We should build in the newest version of Java, but bear in mind some more advanced functions may not work in older versions of Java.
Under estimation of length of various	2	4	Impact reduction	Complete the project production schedule in order and only move on when a previous item is fully

components				complete. Only commit working and tested code so that if time runs short then there is always something to show.

6. Project Schedule

A gantt chart follows this page. Green squares are for development time, blue is for testing or review, red is deadlines.


Month F F F F F F F F F F F F F F F F F F F F F F F F F F F F F M M M M M M M M M M M M M M M M M M M M M M M M M M M
Week 4 4 4 4 4 4 4 5 5 5 5 5 5 5 6 6 6 6 6 6 6 7 7 7 7 7 7 7 8 8 8 8 8 8 8 9 9 9 9 9 9 9 10 10 10 10 10 10 10 11 11 11 11 11 11 11
Day No 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
Day M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S
PROTOTYPE PRESENTATION DUE
GROUP PRESENTATION
CODE SUBMISSION
DOCUMENTATION
All design documentation
GUI Prototype
Visualisation design
Thompson's Algorithm prototype
MODEL
Regex parsing
Checking an input test string for acceptance
Drawing a FSA
Removing Epsilon moves
Determinising
Minimising / Bi-simulation
ANIMATION
Keyframes being passed to visualiser
Animation Speed
Animation Testing
VISUALISATION
Working GLPanel
Drawing States
Highlighting selected states
Drawing Edges
Drag to Move View
Rendering Text for labels
Zooming/Scaling
Drawing curved edges
Arranging the FSA to minimise overlap
Working Swing GUI with all Windows and Panels
Bi-simulation (Split-view) working
UTILITY
Saving and Loading a document
Saving and loading keyframes
Exporting as an image
Regex quick access store