

G52GRP - Interim Group Report
R *Fuzzy* Toolkit

GP12-jmg

Craig Knott (cxk01u)
Luke Hovell (lxh01u)
Nathan Karimian (ndk01u)
George Tretyakov (gxt01u)
Ben Bradley (bxb01u)
With supervision from Jon Garibaldi (jmg)

February 11, 2013

Abstract

This document is the interim group report for the group gp12-jmg, for the G52GRP module at the University of Nottingham, School of Computer Science. It details the progress of our project, a fuzzy logic manipulation tool box, so far; as well as how we have worked together, research we have completed, and how we have dealt with problems. Below are some acronyms that will be used within the document, and their accompanying definitions.

CRAN	The Comprehensive R Network
FIS	Fuzzy Inference System
GUI	Graphical User Interface
IMA	Intelligent Modelling and Analysis
I/O	Input/Out
MF	Membership function

1 Description of the problem to be solved

Fuzzy logic is a form of logic that deals with approximate reasoning, as opposed to fixed, exact values - the variables can have truth values that range from 0, to 1. Fuzzy logic is believed to be better for handling and sorting data, and is an excellent choice for many control system applications due to the way it mimics human control logic.

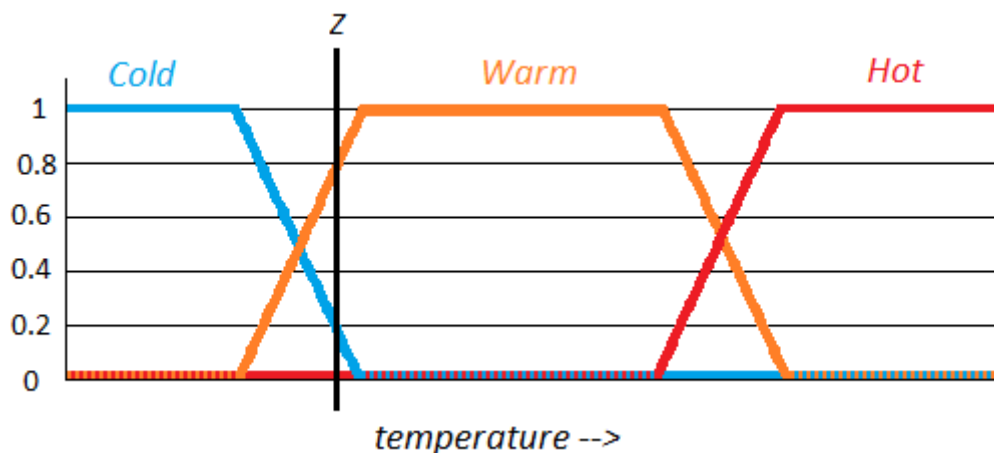
Our project, set forth by the Intelligent Modelling and Analysis group here at the University of Nottingham, was to re-implement the work that they had begun on a Fuzzy-Toolkit, written in the high-level multi-paradigm language, *R*. The aim is for the project to be released on the Comprehensive R Network (CRAN) - a goal that the team have yet to achieve. The tool kit was developed so that the team had a custom framework for working with fuzzy logic (a large part of their research), as opposed to being bound by the restraints of other such suites, for example MATLAB. Unfortunately the functionality of the toolbox is primitive and, as the code has not been updated in a number of years, requires a total overhaul.

The final aim of our project is to rewrite this toolkit into a new, standardised format. This code will then be legible for release as an *R* package on CRAN. The final product of this project will be a full, graphical-user interface for the toolkit, which will communicate with the *R*-Interpreter to create and manage Fuzzy Logic sets. Our system will allow the entry of membership functions (graphs that represent fuzzy data), and Fuzzy Inference Systems (that are a collection of membership functions specified as inputs, outputs and rules). These inference systems can be created on-the-fly (through number input or manipulation of the graph interface), or be read in from a file. The system will also have the functionality to print the inference systems in a user friendly format, display the membership functions as graphs, run through a working demonstration of the system, and allow defuzzification of the functions (which is the process of producing a useful value from a membership function).

2 Background Information and Research

What is Fuzzy Logic?

As mentioned above, fuzzy logic deals with approximate reasoning, with truth values ranging from 0 to 1. This is better for handling and sorting data, as values are not restricted, and can have larger variation. An example of a graphical representation of a fuzzy logic system is shown below. On this example, the x axis represents temperature, and the y axis represents the truth values of the three functions plotted. As the temperature increases, you can see how the truth values of the functions increase and decrease, to give a more accurate result of how warm it is. At point Z , indicated by the black marker, the values of the membership functions cold, warm and hot, are 0.2, 0.8 and 0 respectively. This gives us an expression of the temperature of “slightly cold”, “fairly warm” and “not at all hot”, which is much more precise than a boolean expression of “warm”.



This is the functionality that our system aims to provide, along with other, more advanced tools for evaluation of these graphs.

Research Conducted and Background Information Collected

After our initial research into fuzzy logic, we moved on the programming language R , which was essential to the success of our project. During this time we also made sure to look at any existing fuzzy logic systems, so that we could get a better idea of what sort of functionality would be required of our system. Our supervisor linked us to some resources that he had gathered on the topic of fuzzy logic in MATLAB ¹, which we used to learn about applying fuzzy logic, so that we could better develop a suitable system. Whilst looking for resources on R , one of the group members found a tutorial online² that the R coding team looked at to better understand how to use the language. This gave them the knowledge to understand the initial code supplied by the IMA group, and to allow them to begin coding the new R fuzzy tool kit.

¹ <http://faculty.ksu.edu.sa/abdenmour/Courses/Tutorial%20on%20Fuzzy%20Logic%20using%20MATLAB.pdf>

² www.cyclismo.org/tutorial/R/

After learning about how to use *R* and about fuzzy logic, we looked into currently available products that were similar in functionality to what ours would be. The first we looked at (which we had also used during our learning of *R* and fuzzy logic) was MATLAB. This is a multi-purpose tool that had a small section devoted to the use of fuzzy logic. This was similar to what we would be implementing, but our system would be focusing entirely on fuzzy logic, and had scope to go much more in-depth. The MATLAB fuzzy tool kit was useful so that we could see how a graphical user interface could be applied to fuzzy logic, and to give us possible ideas on how we should implement ours. One advantage that our project would have over MATLAB was that it would be free of charge and open source, whereas MATLAB requires a purchase and does not give out its source code.

Another piece of software that we looked at was XFuzzy 3.0, which was very similar to what we would be creating. The software itself had not been updated since 2003, so was used as more of a reference on other methods used to implement fuzzy logic. This software was command line based, which made the plotting and manipulation of graphs much more difficult for the user. Our project aims to solve this problem by implementing a graphical user interface to the system so that the users can actually visualise their graphs, to make manipulation much easier.

We also looked at the source code for the Version 0.7 Fuzzy Tool kit and Version 1.7 Fuzzy Tool kit, both produced by the IMA group. These would be used, in conjunction with our new specification, to create the brand new fuzzy tool kit. This gave us an idea of the amount of code we would be required to write, how advanced the code was, and supplied us with useful formulae for fuzzy logic, and tips and tricks involving the *R* language.

When researching into suitable technologies and platforms to use, we decided to focus on all major operating systems, as this would be beneficial to the reception of our software. We would write the source code in *R*, which already has a cross platform console, and the graphical user interface in another cross platform language, so that users would not be restricted by their operating systems if they wanted to use our software. We looked into appropriate algorithms and formulae for generating the values we required (membership functions, defuzzification, evaluation values and such), and found that the MATLAB function documentation combined with the 0.7 and 1.7 tool kit source codes gave us a good idea on the best formulae to use, and an explanation of how to use them. Any functions not covered in the 0.7 or 1.7 source codes would be purely down to our judgement as a group. As far as tools go, we decided to use the cross platform *R* console available on CRAN for the implementation of our code, and the adoption of SVN for version control, as both of these were easy to access and use.

3 Requirements Specification

Below we have listed the functional requirements (things that our software must fulfil) and non-functional requirements (restraints placed on our system) of our software system. As our system incorporates both a command line *R* interface, and a Java graphical user interface, some requirements are tailored specifically to one or the other. Any requirement that apply only to the *R* interface will be marked with the letter “R”, and a “G” will be used for the GUI.

3.1 Functional Requirements

1. The users of the system will be able to create Membership Functions, from which they can
 - 1.1 ...create a Gaussian curve
 - 1.1.1 ...specify the parameters of the Gaussian curve
 - 1.2 ...create a double Gaussian curve
 - 1.2.1 ...specify the parameters of the double Gaussian curve
 - 1.3 ...create a trapezoidal graph
 - 1.3.1 ...specify the parameters of the trapezoidal function
 - 1.4 ...create a triangular graph
 - 1.4.1 ...specify the parameters of the triangular function
 - 1.5 ...evaluate the membership function
 - 1.6 ...defuzzify the membership function
 - 1.7 ...plot the membership function (R)
2. The user of the system can manipulate fuzzy inference systems
 - 2.1 The user can create a new FIS, from which they can
 - 2.1.1 ...specify a FIS name
 - 2.1.2 ...specify a FIS Type
 - 2.1.3 ...specify an And Method
 - 2.1.4 ...specify an Or Method
 - 2.1.5 ...specify an Implication Method
 - 2.1.6 ...specify an Aggregation Method
 - 2.1.7 ...specify a Defuzzification Method
 - 2.3 The user will be able to add an input variable, from which they can
 - 2.2.1 ...specify a number of membership functions to add
 - 2.2.2 ...specify the parameters for these membership function
 - 2.3 The user will be able to add an output variable, from which they can
 - 2.3.1 ...specify a number of membership functions to add
 - 2.3.2 ...specify the parameters for these membership function
 - 2.4 The user will be able to add a manipulation rule, from which they can
 - 2.4.1 ...specify a rule (R)
 - 2.4.2 ...use the “Logical Rule Builder” (G)
 - 2.4.3 ...invert any inputs or functions (G)
 - 2.5 The users will be able to evaluate a FIS
 - 2.6 The user will be able to plot a 3D surface of the FIS
3. The user will be able to use file I/O to save and read files
 - 3.1 The user can load in a FIS from a file
 - 3.2 The user can save a FIS to a file

4. The user will be able to print variables in a easily readable format
 - 4.1 The user will be able to print a FIS (R)
 - 4.2 The user will be able to print a rule of a FIS (R)
5. The user can use inbuilt variables to create a new FIS (R)

3.2 Non-Functional Requirements

Availability and Extensibility

The availability of the R Fuzzy Toolbox project will be solely via the internet, and accessible to the public as a package on CRAN. It will be continually kept up-to-date throughout this development year as additions are made to it via a central SVN repository, with the latest version being presented through CRAN.

The system will be extendible and expandable once the final, official release is uploaded onto CRAN. The source code can be requested and any further development beyond the scope of this project can take place. Considering the storage medium is not under our control, availability; such as uptime, will be somewhat further defined by the CRAN online service.

Maintainability

The code will be kept as modular and as readable as possible by the development team, ensuring maintenance and modification (such as the addition of new features) is easy to accomplish by external users. In regards to the project source code, this will be maintained and modified using SVN on a central repository.

Operability and usability

The toolbox is designed to be streamlined and as easy to use as possible due to the wide spectrum of potential users - from novices to advanced. The GUI element of the system will provide an interface to manipulate fuzzy logic in a simplified manner along with the aid of manuals and documentation.

Quality

The system should look, feel, and perform up to the standard and expectation of the user. Any reported faults with the system will be fixed before the package and program's final release.

Resource requirements, constraints and required capacity

In order to use the final version of our program, the user will need to have the Java Virtual Machine (JVM) installed or be familiar with the R console available on CRAN.

It should also be noted that the user will be creating their own data, stored in a .fis file, thus the system the program is being used on will need enough capacity to store this.

Platform Compatibility

The final program will be compatible with any recent version of Windows, MacOS or UNIX.

Security, reliability and robustness

Considering that user input is a potential security vulnerability in any program, input validation and verification is paramount to its safeness. For example, when a function (written in *R*) requires numeric input as one of its arguments, a numeric-based string is obviously required before conversion to pure numerical values. However, without input validation, a user can input a character-based string which, when passed to said *R* function, will result in an error. This could result in the program behaving unexpectedly, passing incorrect or unreliable data or simply crashing. In order to remedy this, all input fields on the GUI will need validation as provided through Java so that data types passed from the Java GUI are consistent with that of the data types required by the *R* functions.

Building on the above point, file I/O security and error checking is somewhat related in that the data written to files, primarily FIS data structures, need to be valid before being written. If incorrect, this will later lead to unreliable data and an inability to read in the file. Because *R* requires an absolute path, and will simply create the directory and file if it does not exist, errors can occur because of permissions, or a non-existent drive being given as an argument. Similarly for reading, if a file is non-existent an error is produced.

Documentation

Both the *R* package and Java classes will have commented code explaining what each segment or block of code does. For example, a function's inputs, outputs and inner workings will all be commented; which should give a clear idea of its purpose and usage. We will be writing JavaDoc comments in our Java code, so that documentation can be automatically generated for us.

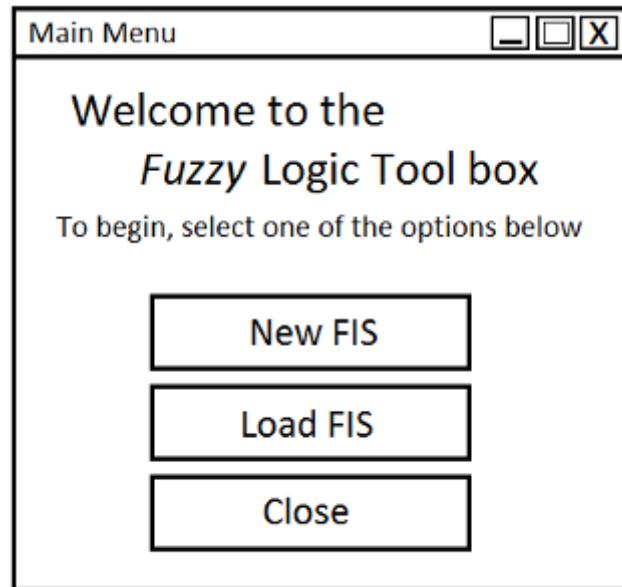
Further documentation will include a manual of the system, which will explain its functionality, and how to use the various elements.

Disaster Recovery

The source files for our project, as well as the documentation is all stored on the University's server (Avon, via H: drive) and are thus backed up to prevent data loss from virtually any threat. When the project is finished and uploaded to CRAN, it will be the responsibility of their server(s) .

4 System Designs

Below are the design of the main interfaces of our system, followed by a quick explanation of each. These designs will be the basis of our GUI for our Java system, and thus are detailed enough with clear indication of what all the elements are to be, and to do.

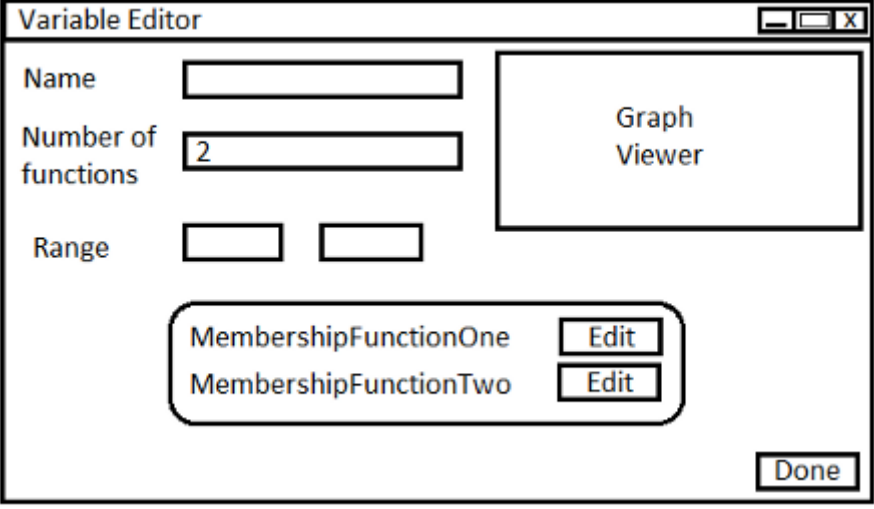


This is the main menu, which is presented to the user upon launching the system. As it was the first thing the user would see, it needed to be easy to use and look at, to give a good first impression. This is why we chose a minimalistic design with little functionality, so the user would not be “thrown in at the deep end”.

The screenshot shows a window titled "FIS Editor" with standard window controls (minimize, maximize, close) in the top right corner. Below the title bar is a menu bar with "File" and "Process" menus. The main content area is divided into several sections:

- FIS Properties:** A rounded rectangle containing five input fields:
 - FIS Name
 - And Method
 - Or Method
 - Imp Method
 - Agg Method
 - Defuzz Method
- Inputs, Outputs, Rules:** Three tabs at the bottom of the main area. The "Inputs" tab is currently selected.
- Inputs Tab Content:** A list box containing the text "{Input Function One}" with an "Edit" button to its right. Below the list box is an "Add New" button.
- Footer:** A text box at the bottom of the "Inputs" tab containing the note: "*Other tabbed panes are the same, except replace the word 'input' with 'output' or 'rule' as required*".

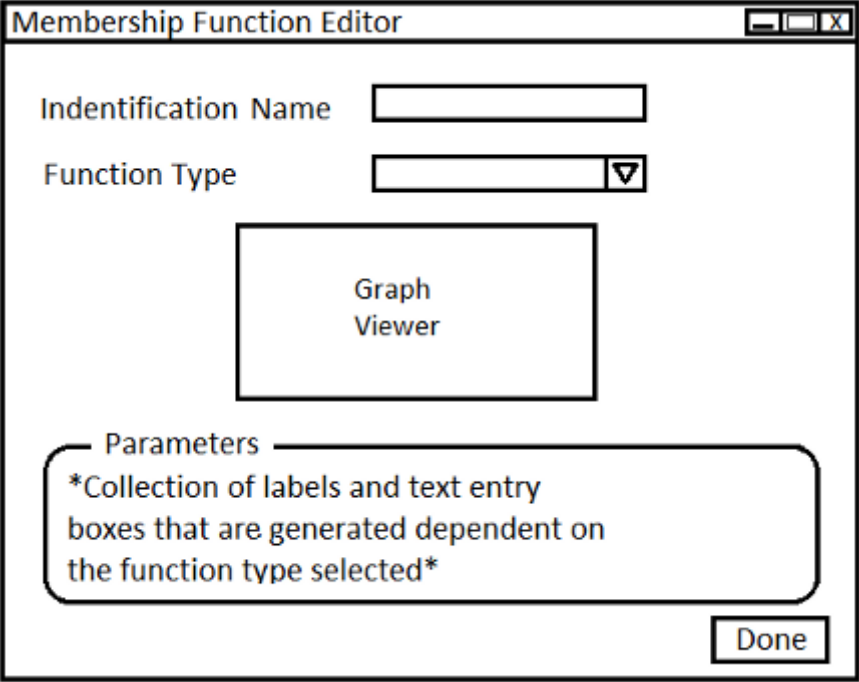
This is the main window of the system (launched after selecting NEW or LOAD from the main menu), and would be what the users would be interacting with the most. It being easy to use was a vital requirement for our project, as the user is only supposed to be limited by their knowledge of fuzzy logic, and not the system they are using. This is why we split up all the elements into tabbed panes, so that the user would not get overwhelmed by the number of items on their screen at once. There was also a menu in place, so that the user could access any extra functionality that they required.



The Variable Editor window is a simple input window with a title bar containing standard window controls. It contains the following elements:

- Name:** A text input field.
- Number of functions:** A text input field containing the value '2'.
- Range:** Two empty text input fields.
- Graph Viewer:** A rectangular area on the right side of the window.
- MembershipFunctionOne:** A label with an **Edit** button next to it.
- MembershipFunctionTwo:** A label with an **Edit** button next to it.
- Done:** A button in the bottom right corner.

The variable editor (launched after selecting NEW on FIS Editor) was a simple input window that allowed the user to specify a number of membership functions to make up a variable.



The Membership Function Editor window is a simple input window with a title bar containing standard window controls. It contains the following elements:

- Identification Name:** A text input field.
- Function Type:** A text input field with a dropdown arrow on the right.
- Graph Viewer:** A rectangular area in the center of the window.
- Parameters:** A label with a line pointing to a large rounded rectangular area containing the text:
Collection of labels and text entry boxes that are generated dependent on the function type selected
- Done:** A button in the bottom right corner.

The membership function editor (launched after selecting EDIT on variable Editor) was a simple input window that allowed the user to specify the dimensions and type of their function, which would then be added to their current FIS.

The screenshot shows a window titled "Rule Editor" with standard window controls (minimize, maximize, close) in the top right corner. The window is divided into three main sections: "Inputs", "Logical connective", and "Outputs".

The "Inputs" section is further divided into "(Input 1)" and "(Input n)". Each input section contains a list box with "Value a", "Value b", and "Value c". To the right of each list box is a vertical scrollbar. Below each list box is an "Invert" checkbox. In the "(Input 1)" section, the "Invert" checkbox is unchecked. In the "(Input n)" section, the "Invert" checkbox is checked.

The "Logical connective" section is located between the input sections and contains two radio buttons: "AND" (which is selected) and "OR".

The "Outputs" section contains a list box with "Value a", "Value b", and "Value c", and a vertical scrollbar. Below the list box is an "Invert" checkbox, which is unchecked.

At the bottom left of the window, there is a "Weight" label followed by a text box containing the number "1". At the bottom right, there is an "Add Rule" button.

The rule editor (launched after selecting NEW on the rule tab of the FIS Editor) was a simple input window that allowed the user to specify a logical construction used to evaluate their FIS. The user could specify the value of the membership functions of the system, how they were logically connected, the weight of the function, and what output this would produce. As this process could potentially be quite confusing the user, we created this graphical interface, so that construction of logical rules was neither daunting nor difficult.

5 Record of key implementation decisions

As a group, we needed to decide on a number of things before production of our system could begin. We discussed which languages would be used to code the software, what operating systems we would support, how we would release the system and what additional software or hardware we would require. A listing of this can be found below, included with reasons as to why each element was chosen over other possibilities.

Main programming language - *R*

In the specification of the project, the language that we were required to write the toolkit in was the high-level multi-paradigm language *R*. The reason that we accepted this and did not contest to it, was that a lot of the code base had already been written in *R*, and rewriting this from scratch in a new language would have been much more work than just learning *R* would have been.

GUI Language - Java

We were also required to write a graphical user interface for our system. As a group, we decided that Java would be the best language to write this in. We attempted to look for a graphical user interface package for *R* and, although we found one, decided that Java would be the better option. The reason for this is that all of us had already learned Java for an entire semester in our first year, and were comfortable using it. We were also required to use it in at least two of our modules in this current semester, meaning we could get even more practice with it. This was opposed to attempting to learn an entirely new language's graphical package, and produce something of a lower quality.

GUI Package - Swing

We decided on using the Java Swing library for our graphical user interface. The reason for this is that some of the group members were already familiar with using this specific package, and most of the group were taking the Graphical User Interfaces (G52GUI) module at the time of beginning the project, which was being taught in Java using the Swing library and thus would teach us the basic of creating the GUI for our system.

Release Pattern - Small, regular, incremental releases over CRAN

We decided that the best way to release the system would be to incrementally release small sections of the code over CRAN, fairly regularly. This way we could assure that our code was up to the standard required of CRAN releases. This also helped to break down the project into smaller segments that made working on it less daunting, as we were only required to complete small sections at a time.

Operating systems supported - Windows, MacOS and UNIX To appeal to wider audience, we will tailor the toolkit to be usable on all three of the main operating systems available at the moment. The *R* Console available on CRAN is already available on a wide variety of UNIX platforms, Windows and MacOS - which means that our source code can be run on any of the main operating systems without us needing to do any extra work. Our GUI will also be usable on these main operating systems, as it will be written in Java, which is already a cross platform language.

Additional Software - Possibly a Java/R communicator

We do not necessarily require any additional software - depending on how we decide to implement the Java to *R* crossover part of our project. We are writing our GUI in Java, and our main code in *R*, which means we need some way of these two languages communicating. If we wish to do this cross over the simple way, we can have our Java code print out the *R* code that the user can run to create the output functions they desire. However, if we choose the more difficult route, and have the Java talk directly to the *R* code, then we may need some middle ground software that will handle this interaction. Writing this ourselves would be far too time consuming and not within the specification of our project. Research is currently under way to investigate a middle ground software.

Additional Hardware - None

As far as additional hardware goes, we decided that we would not need any. The reason for this was that we were developing a piece of software that would be adequate on a regular computer. The system we were creating would not work well as an application on a mobile device, as the system would not be needed “on the go” and wouldn’t gain anything from being mobile (not to mention that inputting data would be made much more problematic).

6 Results of Initial Implementation steps and Prototyping

Fortunately for us, two versions of the *R* Fuzzy Toolkit have already been made by the members of the IMA group. This meant that we had something to base our initial testing on, as we could check that the outputs of our new code is the same as the previous two developed tool kits. There is a version 0.7 kit, and a version 1.7 kit, neither of which are currently available on CRAN, this is due to the IMA group never researching into what an *R* package requires for adequacy of release. The 0.7 Kit used Type - 1 fuzzy logic, and the version 1.7 kit used both type - 1 and type - 2. Our project was only to use type - 1, but this still meant that we had two in-depth resources that we could extract information from, and use to create our new tool kit.

Our initial implementation method was to split down the specification into the core *R* functions. With help from our supervisor, we then ordered this in magnitude of difficulty (to make the initial stages of learning this new language much easier). From this list, we could decide what functions we would complete each week, with a review of our progress every Tuesday in our formal meetings with our supervisor. This has so far proven to be an effective method of delegating the work, and we have managed to keep to our targets each week (with the exception of one, as we were waiting for clarification on the specification - see “7. Problems Encountered So Far” for more details).

Prototyping is a great way to gauge the scope of the project. It involves creating a small working demo of the system, which can be shown to the client for their feedback. Unfortunately, due to the nature of our project, we could not implement a prototype during the initial stages. The beginning of our project involves a lot of programming in *R*, which means a prototype would provide the same functionality as the actual release product, thus defeating the point of making a prototype. When we reach the stage of the project where we can begin programming the graphical user interface; we can begin prototyping the interface, for our client’s opinion. This prototype will allow us to show our client what we have created so far, and get their feedback, allowing us to change the product to tailor to their needs.

Despite our inability to produce a prototype, the nature of our scheduled release plan splits our project into small segments, which can be set as deliverables each week. This means that our supervisor can check our work, to see if the functionality is the same as what they had envisioned. This means that our project will not be completed devoid of any input from our target audience.

7 Problems Encountered So Far

No project is perfect, and ours is no exception. Problems will inevitably appear whilst working on a project, especially when working as a group. Working as a group means that both technical and social problems can occur during the life time of the project, each of which need to be dealt with appropriately and effectively. Fortunately, our group has not encountered too many problems so far, and almost all of the ones that we have encountered have been technical, as opposed to social (which are generally harder to deal with). Discussed below is a list of problems we have encountered so far, how they affected our project, and how we overcame them as a group.

Communication within the group

As we were randomly placed in groups, none of our group members knew each other (well) prior to the start of the project. This meant that we had no way of contacting each other, which was a problem when we needed to organise meetings, or generally discuss things. We rectified this initial problem by setting up a regular meeting (Tuesday Mornings), that we would all attend and could raise any issues we had encountered during the last week. However we soon found that this was an inadequate way of communication, as there was no interaction between one meeting and the next. We decided to also set up a Facebook group for our project, to help rectify this problem. This meant that we could post things to the shared chat area, and have all the other members of the project see, and respond to it. This also allowed us to easily host documents, or share useful links. We also put all our mobile phone numbers on this Facebook group, so that we could get in contact at short notice (especially useful for sending informal apologies for any meetings that we hosted).

Lack of knowledge of fuzzy logic in general

As no one had heard of fuzzy logic prior to the beginning of the project, getting started was difficult. When we began coding the membership function, we were unsure as to what the parameters (given to use in the specification) were to do. This made our initial coding session very confusing and disheartening. We decided to research the functions that we were to implement, to gain a better understanding of what we were to do. We found many resources online that explained the parameters of the functions, and what each function actually did. After this, we could begin coding again with renewed confidence and knowledge.

Lack of knowledge of the *R* language

Much like the previous point, no member of the group had an extensive knowledge of the *R* programming language prior to the projects beginning. This caused many problems in the early stages of our programming sessions. One problem of note was our lack of knowledge of how to update the “*Objects*” we were using in *R*. In most languages, you could pass an Object to a method, edit it, and have the method return the new object. In *R*, we learned, as well as passing the object to the method as a parameter, we had to assign to result to the object for the changes to take place. Thankfully, problems in our *R* code are easy to rectify due the extensive online resources available for *R*. As problems have been arising, we have been working as a team to solve them, and expanding our *R* knowledge together.

Incomplete Specification

Simon Miller, a member of the IMA group, was responsible for sending us a specification of the functions that we needed to write in *R*. This was detailed enough for us to begin work, but we encountered a problem rather early on in the project when two of the functions parameters he specified did not match up. This led us to doubt that the code we had written thus far was correct. We emailed Simon who, after some initial confusion, emailed us a newly revised specification, with this error now resolved.

SVN Commit overwrites

The *R* programming team consisted of three members. Our work ethic was to all meet in the Computer Science Lab (A32), and work on our delegated functions until their completion, helping each other where we could. This worked well for the production of the functions, but when it came to submitting our functions to the code base, some problems occurred. Due to our lack of experience with SVN, half way through one of sessions, we overwrote some code that we had previously written. This was an easy problem to fix, as we just had to SVN update, and commit the code in an ordered fashion, but it did worry us at the time. We made sure that from that point on; we would SVN Update prior to any commits, to make sure that we would not be overwriting any work.

8 Project Time Plan

The time plan laid out below is a list of all the tasks necessary to complete the project, with their associated deadlines attached. The idea of planning out the project in this way was so that the group would have a small deliverable each week, which would help to break down the project into smaller sections, thus easing the work load. Deadlines set out by the module convenor are in bold.

Date	Task to be completed
26/09/12	Provisional Group Allocation
01/10/12	Project Brief Received
02/10/12	Initial Formal Meeting with Supervisor
09/10/12	Allocation of roles within the group
16/10/12	Open/Close issues on Indefero Check in and check out files from SVN repository
30/10/12	Upload project description to Indefero Membership functions of the R Code
01/11/12	Upload all agendas and minutes to Indefero, upload any henceforth as well
02/11/12	Hand In of Project Site
06/11/12	“FIS” functions of the R Code (Excluding <i>EvalFis</i>) Research into packaging of R Preliminary system specification
13/11/12	File Input/Output functions of the R Code
20/11/12	Defuzzification function completed
27/11/12	First Draft of the Interim Report completed
04/12/12	Polishing of the first draft into final draft completed
10/12/12	Hand in of the Interim Report
14/12/12	“View” elements of the GUI Plotting functions of the R code Printing functions of the R code
01/01/13	“Model” of the GUI “Controller” of the GUI
12/02/13	FIS Evaluation function of the R Code
19/02/13	Allocation of sections for final group report
26/02/13	FIS Example Implementation function of the R Code
05/03/13	Implement R/GUI Crossover First Draft of the final group report
12/03/13	Polishing of R and GUI Code
22/03/13	Hand in of final report and software
23/04/13	Begin preparations for the open day and presentation
08/05/13	Open Day
10/05/13	Presentation PROJECT COMPLETION DATE