Fraser Edwards

Refactoring for Open Source Development

Final Report

Imperial College London

Mechanical Engineering

6th June 2014

Abstract

The aim of this project was to improve the aRCPlan program to a stage where it could be released for open-source development.

aRCPlan is a C++ program originally developed by Dr Patrick Leevers (2012) concerned with analysing the rapid crack propagation failure mode within gas or water pressurised pipelines. Rapid crack propagation is characterised by a rapidly progressing crack (100-300 m/s) which can initiate from a sudden impact or existing flaw.

The development of the program was structured into four objectives: refactoring, improvement of inputs and outputs, introduction of a graphic user interface and improvement of the solution method. The first three objectives were completed satisfactorily, although the length of the refactoring phase impacted upon the remainder of the project. Consequently, the solution method was not overhauled as first intended.

Refactoring saw the program reduced into modules wherever possible, making the code easier to read and relate to the underling model (Leevers and Argyrakis, 2010). This should aid future developers as well as making it easier for researchers to introduce changes to sections of the program without having to re-write the entire code.

Drastic improvement of the inputs and outputs included the reading and saving text parameters and production of a range of graphs. Furthermore, files containtaining crack profiles and results are now generated.

The most prominent change from the users perspective was the introduction of the graphical user interface, moving away from a text based system to one where parameters can be edited quickly and without difficulty.

Introducing the interface and plotting capabilities revealed the problems with the underlying model which the final objective was meant to address. Undersired patterns and discontinuities appear in the normalised crack driving force and decompression length graphs. Furthermore, the program produces erratic results at high crack speeds.

The program has now been released for open-source development under the MIT license (§A.5) at the address below:

github.com/FraserEdwards/aRCPlanQt/releases

Table of Contents

1. Introduction

Rapid crack propagation (RCP) is a potentially catastrophic failure mode characterised by a rapid progressing crack (100-300 m/s) which can initiation from a sudden impact or existing flaw in gas or water pressurised pipelines. Although RCP has been researched for a number of years (Massa et al, 1997), interest in the modelling of this phenomenon has renewed with the increased use of polyethylene (PE) pipe in larger diameters (30”) and higher pressures (12 bar) (Palermo et al, 2008).

aRCPlan is a C++ library developed by Dr Patrick Leevers (2012) to transparently analyse RCP using an existing engineering model. The intention is to make aRCPlan available for open-source development, so that researchers may contribute to the model to further its accuracy. The main benefits of open-source development are: little or no cost, continuous development due to multiple contributors and the ability to adapt the software to requirements without invalidating licenses. Prior to release, the program required improvements to both its internal structure and interface which were in places unclear and inelegant.

Figure.1: Rapid crack propagation in small-scale pipe test (Leevers and Morales, 2013)

This report aims to summarise the steps taken during the project as well as explaining the eventual achievements and the final, resulting program. Despite the headings given to each section, their contents overlap significantly due to the nature of the project. Furthermore, source code will not be included in the report. It is, however available at the address below:

github.com/FraserEdwards

1.1. Objectives

aRCPlan was originally written in procedural form, i.e. one function after the next. The most extreme example of this was the central iteration function consisting of 330 lines, moving from test parameters to a full solution. Furthermore, the only functional ways of interacting with the program involved using Terminal[1], shown in Fig.2, making use of the program extremely slow. Changing a single parameter requires multiple key presses simply to navigate the menu before and after entering the value. Moreover, results are presented using the CLI which is incapable of displaying plots or tables.

[1] The Terminal emulator within the OSX operating system providing a command line interface (CLI) or text user interface (TUI).

To address the issues mentioned above, the following objectives were set:

- Refactor the aRCPlan C++ library so that it conforms more closely to open source standards
- Improve the stability of solutions
- Improve inputs and outputs, for example: output to gnuplot[2]
- Introduce a graphical user interface (GUI)

The primary purpose of refactoring is not to achieve efficiency in terms of processing power, speed or even lines of code. It is instead aimed at improving code readability and reduced complexity, hence improving maintainability (Fowler et al, 1999). The final two objectives are concerned with improving the convenience of the program and providing outputs usable in reports.

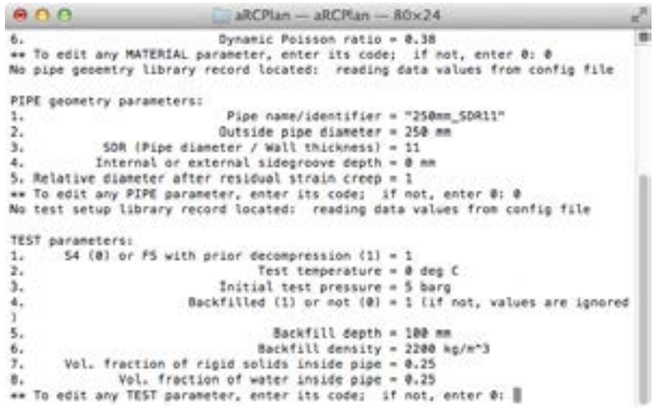


Figure.2: Terminal Interface

Further to the objectives central to the project, gaining a degree of proficiency in C++ formed an additional objective to enable the projects completion.

1.2. Administration

1.2.1. Documentation

A condition of the project was to maintain a logbook containing any information related to the project such as research, objectives, daily work and meeting minutes. In keeping with the digital nature of the project an electronic logbook was maintained using Evernote[3] as opposed to a physical one. Using Evernote it is possible to index and search each record as well as accepting media such as images and hyperlinks. Furthermore, this logbook is available in ‘read-only’ mode online, which allowed supervision between meetings.

[2] A multi platform command line program which can produce 2D and 3D plots of data.

[3] A multi platform application designed for notetaking.

1.2.2. Version Control

As the original program was modified extensively, a version control system was required to ensure a working copy was maintained at all times. The Git (2014) revision control system, implemented through Github[4], was used for this purpose. Git uses a repository to store the current version of a program with a complete history of the changes made, each recorded as a commit, and version tracking. Commits were used for incremental changes and branches for larger changes which would otherwise render the program inoperable during modification. Branches, Fig.3, create a separate set of files from the original program which can be merged once changed have been completed satisfactorily. Alongside revision control, Github provides information on the progress, measured in lines changed and commits.

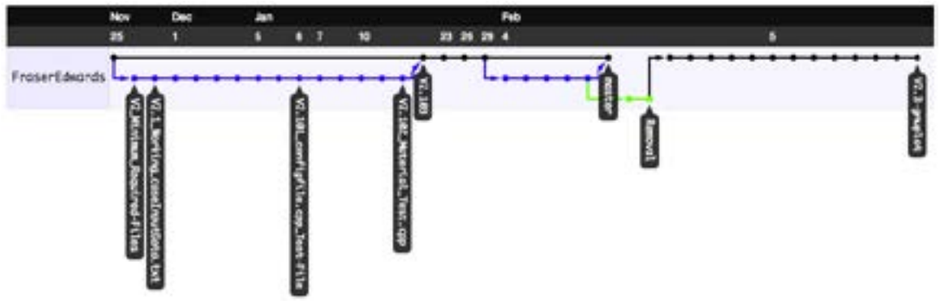


Figure.3: Branch diagram from Github (2014)

1.2.3. Environment

Eclipse was originally intended for use as the Integrated Development Environment (IDE) during the project. However, it does not provide its own debugger, a computer program used to interact and inspect the target program during execution. Eclipse instead provides a front-end for the debugger built into the operating system. Unfortunately, the debugger it seeks to use is not longer supported in OS X Mavericks, the operating system used throughout the project. Consequently, Eclipse was rejected. Xcode was also rejected as an IDE due to its complexity. While comprehensive it is overpowered for development of a CLI. Instead, text editors such as Sublime were used with building, compiling and debugging completed in Terminal. This was achieved through the flag system used to control the default compiler, instructing it to provide debugging information.

To provide the plotting functions integrated as part of the improvement of inputs and outputs, gnuplot was used, chosen for it's comprehensive documentation and extensive use. Gnuplot was originally created to provide scientists and students with a tool for visualising mathematical functions or data. Despite the source code being under copyright, it is freely distributable. In addition to gnuplot, AquaTerm[5] was required to display plots at runtime. Both gnuplot and AquaTerm require installation before use.

To develop the GUI, a number of options were available, including: Xcode, Qt and wxWidgets. Xcode was rejected as it could not provide cross-platform development (only OS X) in addition to the reasons mentioned earlier. Although Qt and wxWidgets both provide cross-platform development, Qt is the more intuitive IDE and is better maintained in terms of

[4] A web-based hosting service for software development using the git revision control system.

[5] AquaTerm is the Mac OS X graphics renderer, allowing command line tools written in a range of languages to display vector graphics, text or images.

support and documentation (tutorials and examples provided). It does however result in bulkier applications as the Qt libraries must be included in the application bundle. Qt (2014) uses a typical designer and two compilers; a Meta Object Compiler[6] and a User Interface Compiler[7] to generate the GUI. Qt's meta object system provides signals and slots for inter-object communications. Signals are generated by the GUI components such as buttons or drop-down boxes with corresponding slots written into the code.

1.2.4. Approach

To complement the learning section of the project, the objectives were tackled in parallel, rather than the order shown in §1.1. This allowed smaller changes to the perimeters of the model to develop understanding of both the language and underlying model.

[6] The Meta Object Compiler enables the processing of Qt's Meta Object System which provides the signals and slots mechanism used for inter-object communication.

[7] The User Interface Compiler reads the user interface form created in the designer, generating a corresponding C++ header file which describes this.

2. Research

2.1. Open Source Development

The aim of the refactoring portion of the project was to simplify aRCPlan's source code so that it could be successfully developed in an open source manner. Open source development, as defined by the Open Source Initiative (2014) must comply with the criteria in Table.1. The benefits of this range from the low to zero cost associated with open-source licenses to the continuous modifications and inspection resulting from multiple contributors. Open-source software also tends to develop at a much faster rate than proprietary software given the same resources (Prencipe et al, 2003).

Table.1: Open source definition criteria (Open Source Initiative, 2014)

Source code	The program must include source code, and must allow distribution in source code as well as compiled form.
Derived works	The license must allow modifications and derived works, distributable under the same terms as the original software.
Integrity of the author's source code	License must explicitly permit distribution of software built from modified source code.
No discrimination against persons, groups or fields of endeavour	License must not discriminate against persons, groups of persons or specific fields of endeavour.
Distribution of license	Rights attached to the program must apply to all to whom the program is redistributed.
License must not be specific to a product	The rights attached to the program must not depend on the program's being part of a particular software distribution.
License must not restrict other software	The license must not place restrictions on other software that is distributed along with the licensed software.
License must be technology neutral	No provision of the license may be predicated on any individual technology or style of interface.

Upon completion of the project, the model described in Table.2 should be used for further development. Whilst these models outline the structure the project and development should take they do not prescribe a style for the code aside from recommending a modular nature. For this, the Google style guide (2014) was used. Where changes were made to the code, the naming convention provided by the style guide was used, with unchanged code remaining in the style it was written. This convention is shown in Table.3 below.

Table.2: Open source development model (Robles, 2004)

Users should be treated as co-developers	Users are treated like co-developers and so they should have access to the source code of the software.
Early releases	The first version of the software should be released as early as possible so as to increase one's changes of finding co-developers early.
Frequent integration	Code changes should be integrated as often as possible so as to avoid the overhead of fixing a large number of bugs at the end of the product life cycle.
Several version	The should be at least two versions of the software: a buggier version with more features and a stable version with fewer features.
High modularisation	The general structure of the software should be modular allowing for parallel development on independent components.
Dynamic decision making structure	There is a need for a decision making structure, whether formal or informal that makes strategic decisions depending on changing user requirements or other factors.

Table.3: Google style guide (2014)

File	Should be all lowercase and can include underscores
Type	Start with a capital letter and have a capital letter for each new word
Variable	All lowercase, with underscores between words
Constant	Use a k followed by mixed case
Function	Mixed case, accessors and mutators match the name of the variable
Namespace	All lower-case, and based on project names and possibly directory structure

The style guide also provides information for structuring comments within the code: function declaration comments describing the use and function definition comments describing the operation. Variables names should, as a rule, be sufficiently descriptive to negate the need for comments describing their function. Whilst some sources (Thereska, 2007) recommend heavily commented code, the style guide (Google, 2014) states that the best code is self-documenting, i.e. getArea gives an indication of the purpose of the function whereas gA does not.

2.2. C++

aRCPlan is written in C++, a general purpose programming language with object-oriented features. The main benefits of C++ are explained in the table below.

Table.4: Main benefits of the C++ language (Stroustrup, 2014)

Encapsulation	Principle of concealing the functional details of a class from the rest of the program.
Inheritance	Ability of one class to inherit members, methods and properties from a parent class.
Polymorphism	Allows different objects to use/respond to methods of the same name.

These object oriented features were used to refactor the original code from a procedural to a more object-oriented structure. Object-oriented programming (OOP) is a programming paradigm in which functions and variables are grouped to form and describe form objects. For example, a box object could have three variables: width, height and depth as well as two member functions: findArea and findVolume. The benefits of OOP largely fulfill the aims of open-source development; providing improved maintainability, improved productivity and higher quality software achieved through a modular structure with objects available for reuse throughout the code.

As mention in §1.1, part of the project involved acquiring sufficient skill in the C++ language to complete the project. A variety of sources were used with C++ for Engineers and Scientists (Bronson, 2012) being the primary source with completed tutorials available through Github at the address below. Alongside this, a combination of websites such as Bjarne Stroustrup's guide to C++ (2012) and minor changes to aRCPlan were used to gain proficiency.

github.com/FraserEdwards/Tutorials

Further to developing programming ability, familiarity with test driven development (TDD) and understanding of the underlying model were required, alongside secondary skills in the IDEs and applications mentioned in §1.2.3.

2.3. Test Driven Development

Freeman and Pryce (2010) describe the underlying theory of test driven development as such:

- Clarifies acceptance criteria for program
- Encourages loosely coupled components
- Detects errors before implementation
- Discourages adding unnecessary features

The fundamental cycle for this is shown in Fig.4.

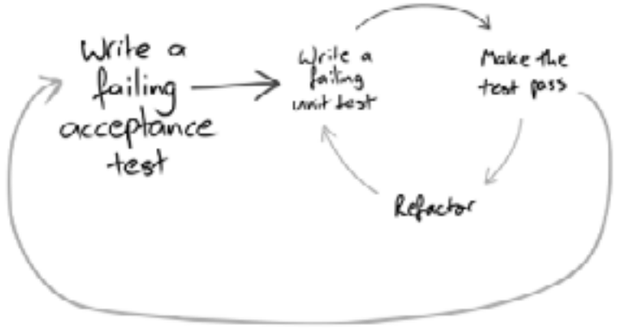


Figure.4: Fundamental TDD cycle (Freeman and Pryce, 2010)

The TDD process involves starting each new feature with a failing acceptance test, a test which when run will show failure of the target of the test. This acceptance test helps define what features are essential, designing the program from the users point of view. In this case the user is both the end user and future researchers who will contribute to the program's development. TDD also helps produce a modular structure, where objects can be run in isolation or swapped without heavy modification to the code.

2.4. aRCPlan

As mentioned previously, refactoring of aRCPlan required a basic understanding of the underlying model (Leevers, 2012) and how the program relates to this. The following sections briefly describe the components of the model. Both Leevers (2012) and Cao (2013) provide more detailed explanations of the model.

2.4.1. Backfill

It is likely that the pipe will be buried following installation. This extern material, backfill, will provide resistance to the radial expansion of the pipe. The engineering model only considers the internal resistance of the backfill due to its density.

2.4.2 Deformation

In order to calculate the crack driving force using the method outlined in the following section, the final crack opening profile must be determined. The pipe wall deformation during fracture can be divided into four distinct parts: radial expansion, lateral bending, torsion and finally vertical bending.

2.4.3. Driving Force

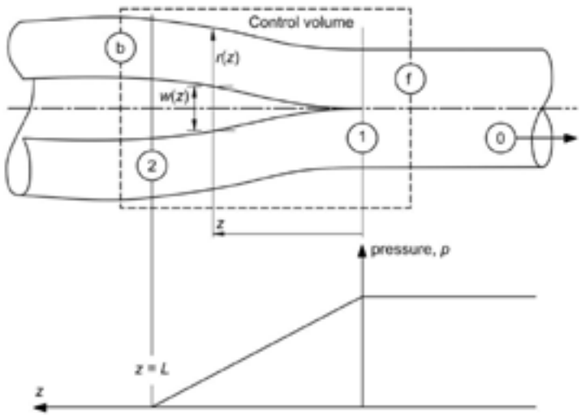


Figure.5: Pipe deformation and pressure distribution during RCP (Leevers, 2012)

Figure.5 depicts a crack propagating from left to right steadily along the axis of the pipe with initial inter pressure. The crack driving force is calculated from the change in strain and kinetic energies during fracture as well as the pressure load upon the crack opening.

2.4.4. Simple Beam Model

A simple beam with elastic foundation, Fig.6, is used to model the crack propagating axially. As shown in Fig.5, the pressure is assumed to decrease linearly to ambient pressure from the crack tip to the outflow point. The crack opening width and decompression length are represented by the beam model displacement and distance between crack tip and outflow point respectively. The outflow point is where the pressure is ambient and the closure point is where the two crack surfaces meet.

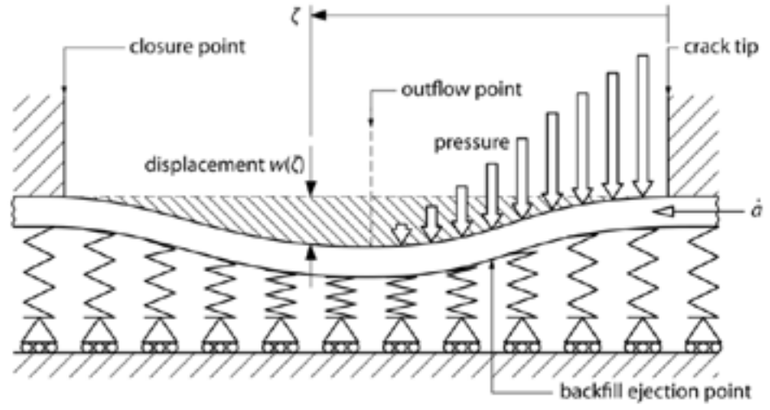


Figure.6: Beam model with elastic foundation (Leevers, 2012)

aRCPlan solves the beam model using a finite-difference method, assuming a decompression length and boundary conditions of zero displacement and gradient at both the crack tip and closure point. This process is repeated with a new decompression length estimated using the outflow model.

2.5. Implementation

Table.5 summarises the main areas describes above and their implementations within the model. The driving force, deformation and backfill components are implemented within the Analytical and Finite Different Solution components.

Table.5: Main constituents of the aRCPlan program (Edwards, 2014)

Model Components	Source File(s)	Description
Analytical Solution	main.cpp AnalyticalProfile.cpp	Produces analytical solution based on 'equivalent beam' model
Outflow Process	OutflowProcess.cpp	Computes pressure as flow discharges through throat area
Finite Difference Solution	main.cpp FDprofile.cpp	Produces finite difference solution also based on 'equivalent beam' model using iterations
Matrix Solver	SymDoubleMatrix.cpp	Solver for symmetrical FD coefficient matrix (used within FD solution)

3. Refactoring

3.1. Original Structure

aRCPlan was originally written in procedural form with the output of one function being fed into the next and so on. As mentioned earlier, the central iteration function totalled 330 lines taking test parameters as arguments and returning a complete solution, making the program extremely difficult to understand and modify. Furthermore, the functions within the original program were tightly linked and as a result inflexible with little generalisation. The primary concern of refactoring is not to improve the efficiency of the program but to improve nonfunctional attributes of the code such as the ease of reading and maintaining it.

One method to view the relationships between classes, shown in Table.6, and objects with a program is a unified modelling language (UML) map. UML is designed to present a standardised method of viewing the design of a system/program and achieves this by graphically representing relationships. The most basic of which are: associations and inheritance (Bronson, 2010). Associations are typically accompanied by phases such as: "consists of", "is related to" or "works for" and values to describe the multiplicity of the relationship. Multiplicity describes the number of one class in relation to any other ranging from a one to one relationship to zero to any positive integer. As can be seen in the UML map of the original program attached in §A.1, none of the original classes are inherited from each other and only two are truly related using the strict definitions for UML (SymDoubleMatrix and FDprofile) outside of the main source file.

Table.6: Original Classes

Class	Status (Complete)	Description
AnalyticalProfile	Active	Analytical solution of 'equivalent beam' model
BeamModel	Inactive/Incomplete	Equivalent beam' model, intended to supersede the computeG function in the main file
ConfigFile	Active	Reads caseInputData.txt to Control, Material, Pipe and TestSetup classes
Constants	Active	Provides constants throughout the program
Control	Active	Contains and allows user to modify Control parameters thought Terminal Interface
FDprofile	Active	Finite-difference solution of 'equivalent beam' model
GaDotSolution	Active/Incomplete	Copy of GaDotSolutionFD
GaDotSolutionAnalytical	Active/Incomplete	Intended to output analytical solution to text and comma delimited files
GaDotSolutionFD	Active/Incomplete	Intended to output finite-difference solution to text and comma delimited files
GSolution	Inactive/Incomplete	Empty
Material	Active	Contains and allows user to modify Material parameters throught Terminal Interface
OutflowProcess	Active	Computes pressure as flow discharges through throat area.
Pipe	Active	Contains and allows user to modify Pipe parameters thought Terminal Interface
SymDoubleMatrix	Active	Solver for symmetrical FD coefficient matrix (used within FDprofile)
TestSetup	Active	Contains and allows user to modify TestSetup parameters thought Terminal Interface
main	Active	Orchestrates calculation at higher level, contains iteration function

3.2. Reducing

As can be seen in Table.6, aRCPlan was left mid-development. As a result, the first task undertaken in refactoring was the removal of incomplete or uncalled functions. These were stored in a reference folder as they suggested directions in which to move the program, for example: the secondary OutflowProcess source file. The beam model class was of particular interest as it was intended to replace the computeG function within the main source file.

As well as the 'skeleton' classes and functions, the analytical solution method was removed due to its fundamental unsoundness (Leevers and Argyrakis, 2010), leaving behind the slots for this solution method should a valid method become available. Consequently a number of classes dependent on the analytical solution and their files were removed: the GaDotSolutionAnalytical source and header files.

3.3. Modularising

On inspection of the underlying model it became apparent that the calculation could be split into a number of modules corresponding to the components identified in t§2.4. The following elements were moved in to their own classes: Backfill, Decompression (downstream outflow) and Creep (calculates diameter as a result of residual strain contraction). A class called FracMech (Fracture Mechanics) was also created, responsible for calculating the crack driving forces This approach drastically simplified the iteration function within the main source file. Most significantly, it provides the opportunity for more detailed, accurate models to be incorporated at a later date without modifying code central to the program.

3.4. Refactoring

Following the preparatory work mentioned above, the refactoring began in earnest. This involved reducing large functions into multiple smaller ones with a particular focus, for example: one function to open a file, another to write column titles and another to fill the file with results. This resulted in a greater number of functions, lines of code (Fig.7) and files (source and header). The most significant reduction occurred in the main source file which reduced from 452 to 19 lines of code; the removed code relocated into multiple classes.

Figure.7: Code additions/deletions for aRCPlan (Github, 2014)

The end result of this refactoring is shown in the UML map attached in §A.2. The structure of the program changed significantly, moving to a much more object-oriented structure with instances passed and returned from functions. Furthermore, inheritance was used to simplify the program where possible. The primary example of inheritance within the program was the creation of the Parameters class which inherits functions and collects values from the Control, Material, Pipe and TestSetup classes, grouping these into a single object to be passed between functions.

Table.7: Final terminal version class

Class	Description
Backfill	Contains the backfill component of the model
BeamModel	Equivalent beam' model, supersedes computeG function in the main file
ConfigFile	Reads caseInputData.txt to Control, Material, Geometry and TestSetup classes
Constants	Provides constants throughout the program
Control	Contains and allows user to modify Control parameters thought Terminal Interface
Creep	Contains the creep component of the model
Decompression	Contains the decompression component of the model
FDprofile	Finite-difference solution of 'equivalent beam' model
Filepath	Finds and stores the location of the application
FracMech	Calculates the crack driving forces
Geometry	Copy of GaDotSolutionFD
Interface	Contains and allows user to modify Interface parameters as well as controlling the Terminal interface
Material	Contains and allows user to modify Material parameters throught Terminal Interface
OutflowProcess	Computes pressure as flow discharges through throat area.
Parameters	Groups the Control, Material, Geometry and TestSetup classes in to one object to efficiently pass between functions
Plot	Class providing high level plotting control through gnuplot interface library
Results	Creates a comma separated value file storing the results
Simulation	Now the overall iteration function concerned with the independent variable
Solution	Class storing the solution for access by other functions
SymDoubleMatrix	Solver for symmetrical FD coefficient matrix (used within FDprofile)
TestSetup	Contains and allows user to modify TestSetup parameters thought Terminal Interface
main	Orchestrates calculation at higher level, contains iteration function

4. Inputs and Outputs

4.1. Initial

4.1.1. Input

The original program was controlled exclusively through Terminal, reading default values from libraries stored in the header files of the material, pipe, control and testSetup classes. Although a method to load and assign values from a file existed within the program, the file it was intended to process was empty. Parameters were edited through Terminal using a numerical referencing system, shown earlier in Fig.2, resulting in an extremely slow interface.

4.1.2. Output

The output of the original program consisted of a set of crack profile displacement values and accompanying results without headers displayed through Terminal, shown in Fig.8. A number of files were created and headers written but no results entered into these.

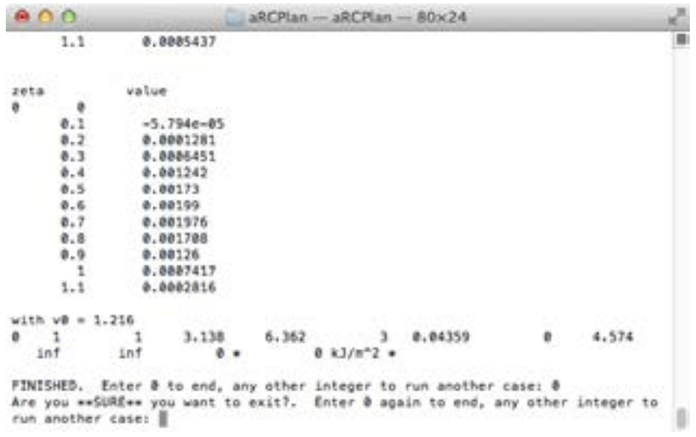


Figure.8: Original Terminal view of results

4.2. Final

4.2.1. Input

The first task to improve the inputs was to reverse engineer the file the configFile class was built to read. The resulting file can be found in §A.6 and represents the quickest way to run a test with a given set of parameters as the Terminal interface can be skipped through.

Unfortunately, the Terminal limited any improvement which could be made to the interface.

4.2.2. Output

In contrast to the minor changes made to the inputs of aRCPlan, the outputs were improved drastically, falling into two categories: Terminal related and otherwise. The Terminal output was reduced from showing multiple crack profile iterations to a first guess and final profile. The solutions shown through Terminal were modified so that they were laid out vertically with headings preceding each value to improve legibility and understanding of each term, shown in Fig.9.

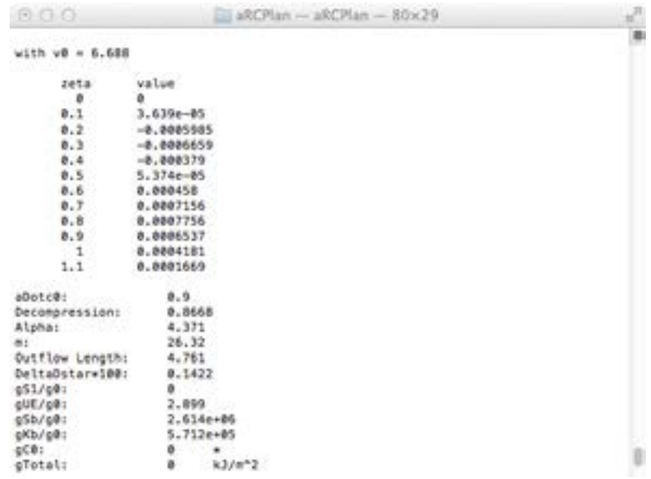


Figure.9: Final terminal output showing crack profile and results

The multiple results files created originally were reduced to a singular text file which was formed into a table format using spacing. However, this did not present a particularly elegant solution and would be difficult to import into other programs. Hence, the output file was changed to a comma separated value file (.csv). Although when first opened the formatting is not visually appealing with headers shrunk to maintain a constant column size, this format is simple to open in programs such as Matlab and Excel.

The most significant improvement to aRCPlan was the introduction of plotting capability, achieved through gnuplot. To interface between aRCPlan and gnuplot, a C++ library maintained by Jeremy Conlin (2009) was used. The library ‘pipes’ commands from aRCPlan to the gnuplot interface to produce the plots shown below. Plots were created for both crack profiles and the final results (Fig.10) with each one saved to appropriate locations. Although simple and effective, using the C++ library mentioned above does limit the program. The gnuplot graphs are opened as temporary files, without a method to close these. Each operating system, OS X and Windows, has a limited number of these temporary files that it can maintain. As a result, the number of speed iterations is limited to approximately 50, if the crack profiles are plotted.

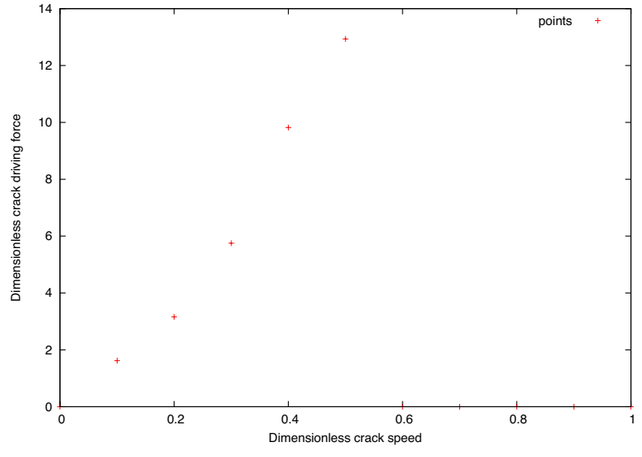


Figure.10: Normalised crack driving force plot produced by aRCPlan through gnuplot

5.1. Interface

5.1.1. Parameters

The interface allows the user to edit all input parameters in the model across four categories: Pipe Material, Pipe Geometry, Test Setup and Control. Multiple parameters can be edited between simulations.with standard pipes materials and geometries are accessed via dropdown boxes.

5.1.2. Operation

aRCPlan can be run in two distinct modes: single and range. As the name suggests, single mode runs a single simulation with the given parameters. Range mode runs multiple simulations with a given independent variable: Crack Speed, Initial Pressure or Temperature. This range can be narrowed to focus on certain behaviour in the model. Although not shown here, it is possible to load and save the inputs used to a text file using the toolbar.

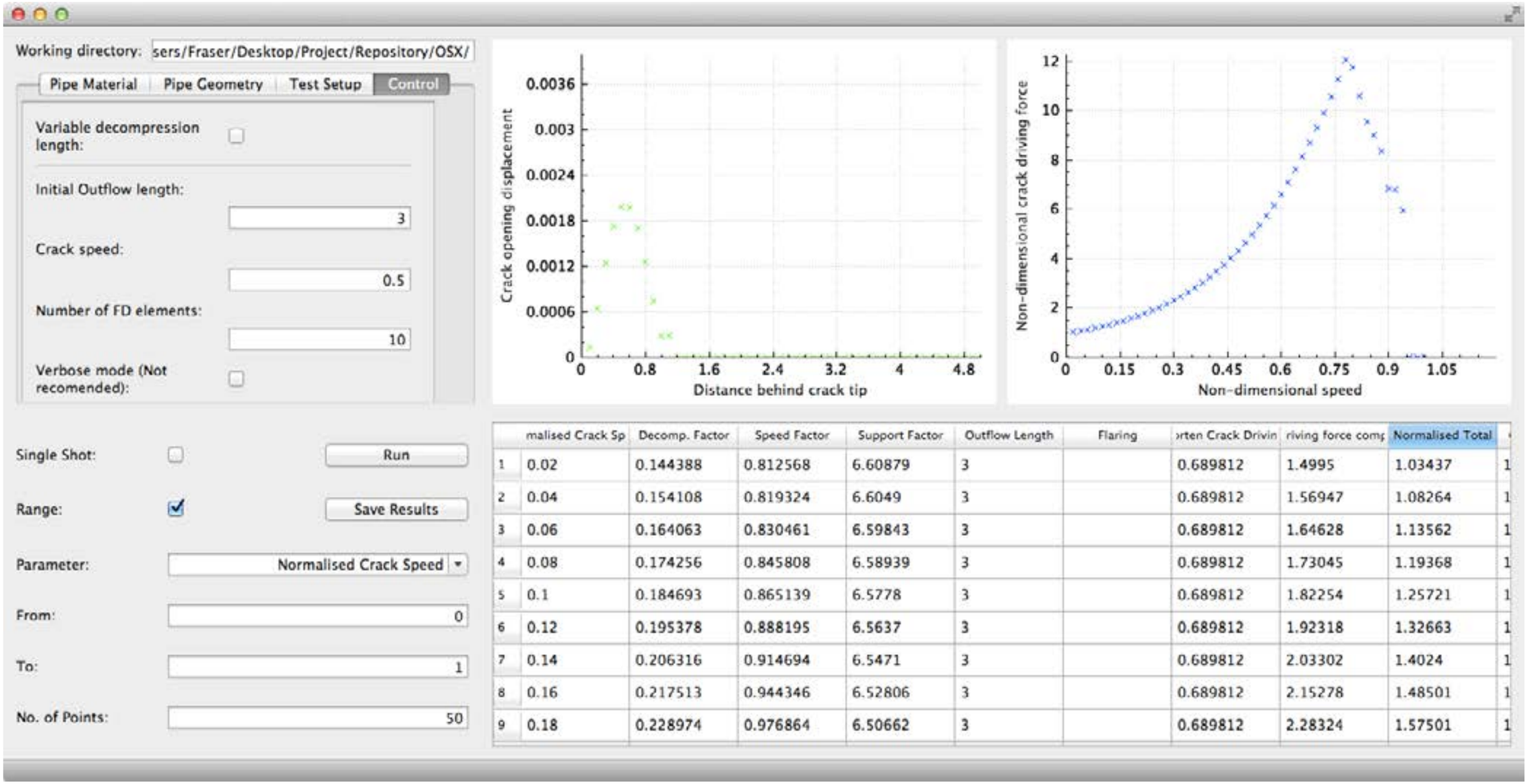
5.1.3. Crack Profile

Each solution presents a crack profile (green for converged and red for unconverged) . These profiles are saved when accessed by the user in the interface rather than upon creation as this would result in potentially thousands of plots.

5.1.4. Results Profile

A range of values are plotted for each solution, from the final support factor to the non-dimensionalised crack force. Each of these is plotted against the independent variable.

Plots are controlled using the table: rows correspond to crack profiles, columns to results.



	malised Crack Sp	Decomp. Factor	Speed Factor	Support Factor	Outflow Length	Flaring	orten Crack Drivin	riving force comp	Normalised Total	
1	0.02	0.144388	0.812568	6.60879	3		0.689812	1.4995	1.03437	1
2	0.04	0.154108	0.819324	6.6049	3		0.689812	1.56947	1.08264	1
3	0.06	0.164063	0.830461	6.59843	3		0.689812	1.64628	1.13562	1
4	0.08	0.174256	0.845808	6.58939	3		0.689812	1.73045	1.19368	1
5	0.1	0.184693	0.865139	6.5778	3		0.689812	1.82254	1.25721	1
6	0.12	0.195378	0.888195	6.5637	3		0.689812	1.92318	1.32663	1
7	0.14	0.206316	0.914694	6.5471	3		0.689812	2.03302	1.4024	1
8	0.16	0.217513	0.944346	6.52806	3		0.689812	2.15278	1.48501	1
9	0.18	0.228974	0.976864	6.50662	3		0.689812	2.28324	1.57501	1

Figure.11: Graphical User Interface

5.1.5. Table

The results table is generated following the simulation, displaying the same results as the original aRCPlan program. Further to this, it allows control of the plots above. At present the table is read-only and hence cannot be edited. Once the simulation has been run, files containing crack profiles and results can be generated using the ‘save results’ button.

5.2. Controls

The controls to the left of the main window provide an interface between the input parameters and the calculation which is started on selection of the ‘run’ button. Using the GUI as an interface removes the need to store the parameters as a global object. These parameters can be edited by the user prior to running the calculation.

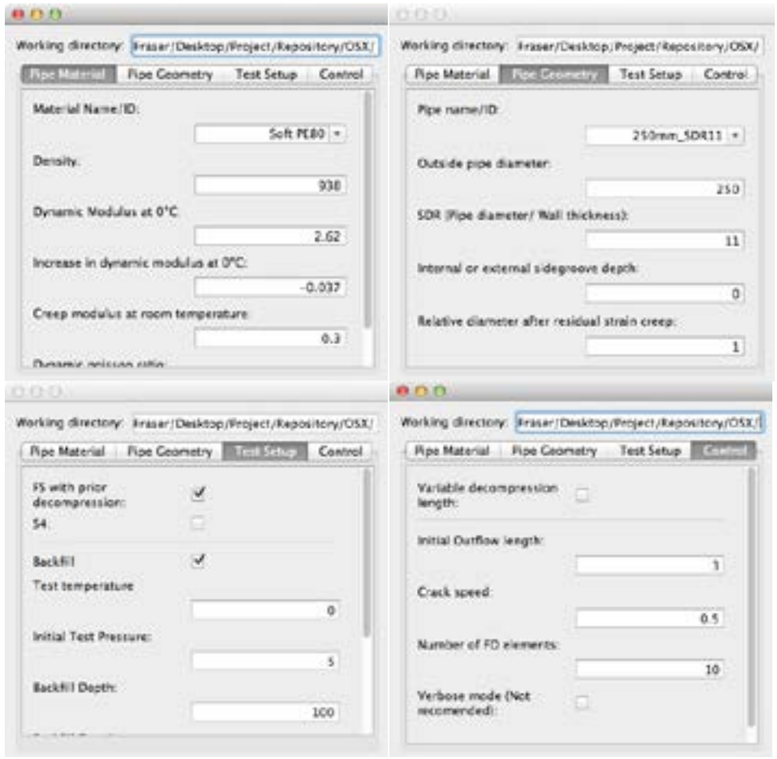


Figure.12: Control windows side by side

Furthermore, “caseInputData.txt” can be loaded and crucially saved, allowing the user to run specific cases.

5.3. Outputs

The GUI displays three outputs once the calculation for a given case has finished running: a table, plot of the crack profile and plots of the eventual results. The table and plots display the same results displayed in the original program. The result table, attached in §A.6, is used to control the plots displayed above it: the column determining which results plot will be displayed and the row determining which crack profile is displayed. The GUI provides the options to save results from the case.

5.4. Refactoring

As a result of the significant changes created while implementing the GUI, further refactoring was required. The Interface class was superseded by the GUI classes built by Qt and hence was removed, along with all references to it.

Solution was modified to become a global object, allowing it to be accessed from any function within the GUI. This was essential to producing the plots and results files. An alternative method to this would involve storing the data in the table displayed in the bottom right corner of the GUI. The data would then be read from the table to generate the plots and results files when required.

The Filelocation class was heavily modified and expanded to increase its capabilities from only providing the location of the program within the computers directory to writing the results and test parameter files. Eventually it was expanded to write the log file used in the following section.

6. Solution Method

Although improvement of the solution method was intended to precede the development of the GUI, this was delayed as the issues reported at the beginning of the project could not be found. Wherever the program produced spurious results or crashed during refactoring, this was due to a mistake made during the changes rather than a problem with the underlying model.

The reported issues only became apparent upon the implementation of the GUI, specifically when the number of iterations increased over fifty. With a high number of iterations and a constant decompression length, a distinct pattern forms.

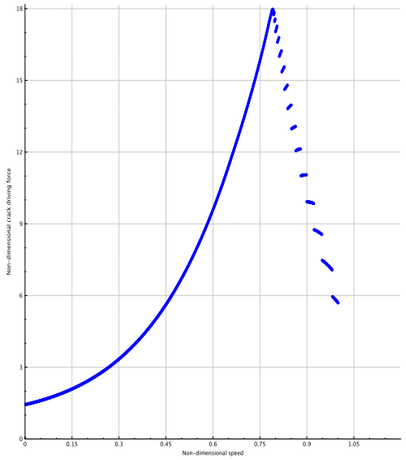


Figure.13: Plot showing patterns

With a variable decompression length, this pattern disappears. However, the outflow length shows a distinct discontinuity in Fig.14. Furthermore, the normalised crack driving force behaves unpredictably with spurious results above an approximate normalised crack speed of 0.6.

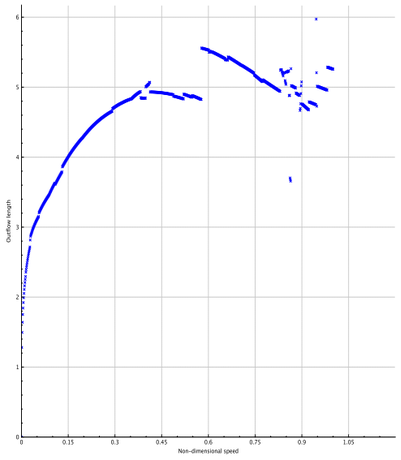


Figure.14: Plot showing discontinuity in decompression length.

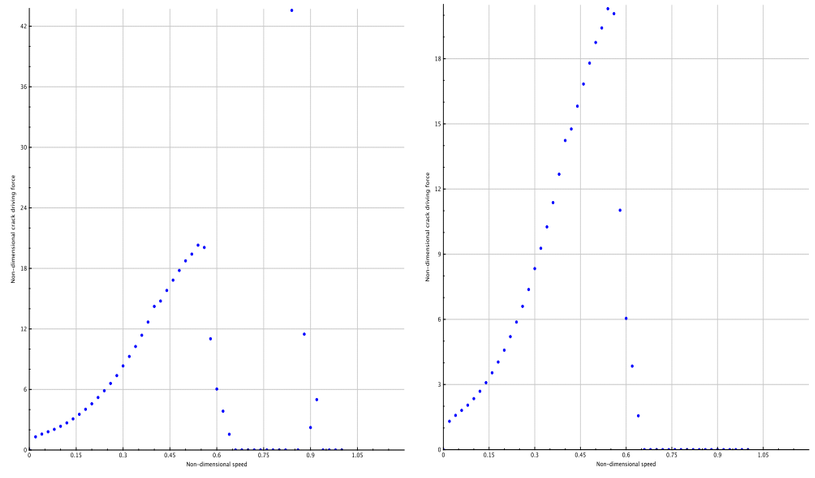


Figure.15: Plots showing different plots for same parameters

To attempt to understand and pinpoint the source of these phenomena, the File class was modified to produce a log file to which each component of the model reports. This log file, attached in §A.7, was used to locate the source of the error. By consulting the log file, it became apparent the discontinuity present in the decompression length plot was the result of the outflow process model. Unfortunately, the project concluded without finding the fundamental sources of each error. Consequently, this has formed a component of the recommended future work outlined in §9.3.

7. Testing

Testing throughout the project took place on two broad levels: accuracy of overall results and behaviour of individual classes or functions.

7.1. Accuracy

At the beginning of the project, a full set of source code was copied from the Github repository and isolated in a separate folder. This original version was used to provide two full sets of results stored in a text file which were available for reference at any time. Each set of results used different parameters to ensure that all areas of the code were tested. Furthermore, maintaining an original version allowed the use of more intimate debugging techniques, such as using breakpoints of outputs to the CLI to compare values within the program at run time.

Following each major change, simulations were carried out with the test parameters and compared to the results from the original to detect any errors.

7.2. Unit Testing

Initially unit-testing was intended for use throughout the project, with tests created for the following classes: ConfigFile, Control, Decompression, Material, Pipe and Test Setup. However, it quickly became apparent that generating these tests drastically reduced the rate of progress in the project. Furthermore, the product of the majority of functions and classes were easy to check, for example: the results functions generating a comma separated file.

Although using the program in various modes and verifying the results provides a good representation of how functional the program is, it does not guarantee that the program is bug free. This will emerge as the program enters continued use and open-source development with contributors identifying potential issues. Despite unit testing not being of significant utility in this project, it is an effective tool for constructing software where the results are not easy to visualise such as the transmission of data or the behaviour of a section of the model across a range of values. In the future, the behaviour of models such as the creep deformation could be investigated using unit tests.

7.3. User Testing

Due to a lack of time towards the end of the project as well as its rapid development, the program was not heavily user tested. However, a notable indicator is the discovery of the underlying issue with the model described previously. This problem became apparent due to the improved outputs and ease of use of the final program allowing parameters to be changed rapidly between simulations.

8. Release

8.1. License

Whilst Open source software naturally suggests a lack of a license, generally this is not true. A wide range of OSS licenses are available, the most common of which is the GNU General Public License. The permitted and required actions are shown below alongside those for the MIT license, considered to be the shortest and simplest license. The MIT license effectively allows any use, including selling, or modification to the code with proper attribution.

Table.8: Requirements of license (Github, 2014)

	GPL	MIT
Required	• Disclose Source • License and copyright notice • State changes	• License and copyright notice
Permitted	• Commercial Use • Distribution • Modification • Patent Grant • Private Use	• Commercial use • Distribution • Modification • Private Use • Sublicensing
Forbidden	• Hold Liable • Sublicensing	• Hold liable

The MIT license was chosen for its simplicity and permissiveness and is attached in §A.5. However, the license can be changed prior to release for open source development depending on the concerns of Dr Patrick Leevers.

8.1. Location

The program is available at the address below in release form. This release contains both a working app bundle and the source code. At present, aRCPlan is only available for OS X although the program could be easily deployed on to Windows if the project was built using Qt installed in a Windows environment.

github.com/FraserEdwards/aRCPlanQt/releases

8.2. Versions

A version system was outlined in the plan report (Edwards, 2013) with major version numbers corresponding to the inclusion of features or completion of major changes. This version system performed well up until multiple development activities overlapped at which point this system broke down. As a result, the system will be kept internal to the project with the release at the above link recorded as V.1.

9. Discussion

9.1 Progression

The original Gantt chart (Edwards, 2013) was created at the beginning of the project in an aim to provide structure and guarantee completion of the objectives. In this initial chart, the tasks roughly followed on from each other. In reality most of these tasks ran in parallel, a consequence of the approach taken to the project to work inwards from the perimeter of the program to its core as proficiency in C++ and understanding of the underlying model was gained. Moreover, changes to the inputs, outputs and introduction of a GUI required further refactoring to clean up the code.

The reality of the project is reflected in the final Gantt chart attached in §A.8. As can be seen, the buffer time allocated originally was fully utilised, with the investigation of the solution method occurring entirely within this time.

Figure.16: Github graphs showing contributions

From the final Gantt chart and graphs produced by Github, Fig.16 , it is obvious that too much time was spent learning C++, which would have been better spent making minor changes to the program as these proved to be more effective at improving proficiency. This apprehension to delve into the code was a result of intimidation by the large size of the program and how unclear some sections of the code were. Fig.16 also demonstrates how the rate of progress increases as the project has advanced; the troughs as a result of external coursework commitments.

The consequence of this over-run in the learning process was a reduction in the time available to improve and investigate the solution method, which became the final objective as the reported problems could not be found prior to developing the GUI.

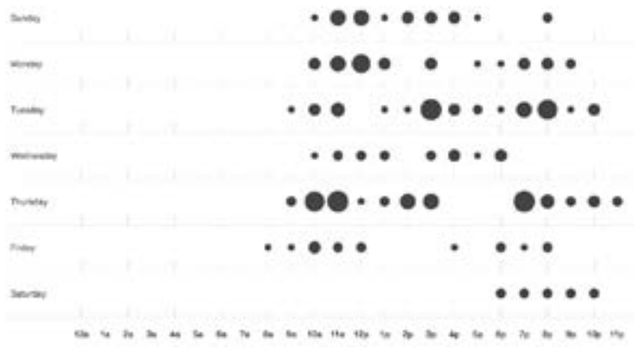


Figure.17: Commits with respect to day and time

The plan report (Edwards, 2014) included a timetable displaying the time allocated during a standard week to the project. The timetable concentrated the work into the weekdays, specifically Tuesdays and Wednesdays. As can be seen in Fig.17, the work was distributed evenly across the majority of the week. This deviation from the original plan is the result of external commitments.

9.2. Critique

The refactoring of the program went relatively smoothly once initial apprehension to editing the code was overcome. aRCPlan was significantly restructured and broken down into smaller functions which are easy to modify. However, it could be argued that constructing the program from new may have resulted in better structured code. Most of the changes made were incremental to maintain functionality and accuracy of the model at all times. Starting afresh would rid the program of some of its peculiarities. The improvement of the inputs, outputs and GUI has been significant, most notably allowing problems with the central model to be visualised as in Fig.13.

Although difficult to quantify, the GUI appears to have improved the user friendliness greatly. Multiple parameters can be changed quickly in between simulations, including the ability to focus in on a certain area of behaviour within the range mode. Furthermore, the GUI allows the graphs to be seen within the program rather than requiring the user to open the corresponding folder to view these. This simple change allows the user to check the validity of the solutions produced without exiting the program.

Due to the learning process occuring alongside the development of the aRCPlan program, some of the code may not be optimised.

9.3. Further Work

This following section deals with further work which could be carried out at a later date external to the underlying model.

A recommended starting point would involve improving the logic built into a lot of the functions to ensure that they behave correctly under all conditions, with a focus on error handling. A similar task would involve creating a warning system to prevent the user entering combinations of parameters which would result in the solver method crashing due to an excessively large array. Although crashing the program is difficult to achieve, preventing this occurring at all would be worthwhile.

The central iteration function is still substantial and could benefit from further refactoring with a focus on achieving a solution with fewer iterations. This would have to be carried out with care as the central model is so tightly knit that circular header references can be created which will prevent the program compiling.

Now that it has been introduced, the GUI could be improved without much effort. At present, none of the elements scale and as a consequence the window size is fixed. The GUI can also be used to implement changes within the source code such as increasing the number of variables which the simulation can iterate through.

During development of the GUI, a ‘connect’ method within Qt was attempted. This method allows any class to provide a signal with a matching slot function. This would allow the source file concerned with the gui to be reduced drastically as at present it controls the entire GUI.

Finally, the results, crack and log file functions could be split into individual classes inheriting from the file class to simplify the code further.

10. Conclusion

In summary, the project has completed three of the four main objectives specified at its initation (§1.1), moving aRCPLan from a Terminal based program with minimal interaction to a functional GUI with report quality outputs. Although the final objective has not been fulfilled, significant progress has been made into determining the underlying source of the errors in the underlying system.

aRCPlan is now capable of reading and saving test parameters from and to a text file, allowing the user to re-run specific cases without difficulty. Furthermore, it now creates a number of output files storing the simulation results and crack profiles.

Plotting functionality has been integrated into aRCPlan, displaying the graphs to the user within the GUI and saving to the relevant directory. These plots along with the increased convenience provided by the GUI have uncovered the issues with the underlying model which formed the fourth objective which was not completed.

Recommendations for further work include the checking of the logic structure in the code with the option to introduce exception handling and further refactoring.

As a result of the projects completion, aRCPlan has been released with the MIT license (§A.5) through the address below to allow for open source development.

github.com/FraserEdwards/aRCPlanQt/releases

11. References

Bronson, G. R. (2010) C++ for Engineers and Scientists. [Online] Available from: 3rd .

Cao, J. (2013) Refactoring Engineering Software in C++ for Open-Source Development. Masters. Imperial College London.

Conlin, J. (2009) C++ Interface to Gnuplot via POSIX pipes (1.0) [Freeware] .

Edwards, F. (2014) Project Progress Report: Refactoring for Open-Source Development. Report edition.Imperial College London, London.

Edwards, F. (2013) Project Plan Report: Refactoring for Open-Source Development. Report edition.Imperial College London, London.

Fowler, M., Beck, K., Brant, J., Opdyke, V. & Roberts, D. (1999) Refactoring: Improving the Design of Existing Code. United States, Addison Wesley Longman Inc.

Freeman, S. & Pryce, N. (2010) Growing Object-Oriented Software, Guided by Tests. Addison-WEsley Signature Series. USA, Pearson Education.

Git. (2014) About. [Online] Available from: http://git-scm.com/about [Accessed 31/05/2014].

Github. (2014) Choosing a license. [Online] Available from: http://choosealicense.com [Accessed 31/05/2014].

Github. (2014) Fraser Edwards Profile. [Online] Available from: https://github.com/FraserEdwards [Accessed 31/05/2014].

Google. (2014) Google C++ Style Guide. [Online] Available from: http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml [Accessed 31/05/2014].

Leevers, P. (2012) An Engineering Model for Rapid Crack Propagation along Fluid Pressurized Plastic Pipe. Engineering Fracture Mechanics. 96, 539-557.

Leevers, P. & Argyrakis, C. (2010) A Basic Model of Rapid Crack Propagation in Pressurised Plastic Pipe. UK, Imperial College London.

Massa, F., Piques, R & Laurent, A. (1997) Rapid crack propagation in polyethylene pipe: combined effect of strain rate and temperature on fracture toughness. Journal of Materials Science. 32, 6583-6587.

Morales, A. G. & Leevers, P. (2013) Residual stress effect on rapid crack propagation in polyethylene pipes. [Online] Available from: http://www.4spepro.org/view.php?article=004641-2012-12-22 [Accessed 31/05/2014].

Open Source Initiative. (2014) The Open Source Definition. [Online] Available from: http://opensource.org/osd [Accessed 31/05/2014].

Palermo, G., Michie, W. J. & Chang, D. (2008) Increasing Importance of Rapid Propagation (RCP) for Gas Piping Applications - Industry Status. Pipeline & Gas Journal. 235 (12), .

Prencipe, A., Davies, A. & Hobday, M. (2003) The Business of Systems Integration. United Kingdom, Oxford University Press.

Qt. (2014) Developer Guides. [Online] Available from: http://qt-project.org/wiki/developer-guides [Accessed 31/05/2014].

Robles, G. (2004)
A Software Engineering approach to Libre Software Open Source Annual. , .

Stroustrup, B. (2014) The C++ Programming Language. [Online] Available from: http://www.stroustrup.com/C++.html [Accessed 31/05/2014].

Stroustrup, B. (2012) Learning and Teaching C++. [Online] Available from: http://www.stroustrup.com/C++.html#learning [Accessed 06/02/2014].

The Saylor Foundation. (2013)
Advantages and Disadvantages of Object-Oriented Programming (OOP) [Online] Available from: http://www.saylor.org/site/wp-content/uploads/2013/02/CS101-2.1.2-AdvantagesDisadvantagesOfOOP-FINAL.pdf [Accessed 31/05/2014].

Thereska, E. (2007) C Coding Standard. [Online] Available from: http://users.ece.cmu.edu/~eno/coding/CCodingStandard.html#cstas [Accessed 31/05/2014].

A. Appendices

A.1. UML: Original

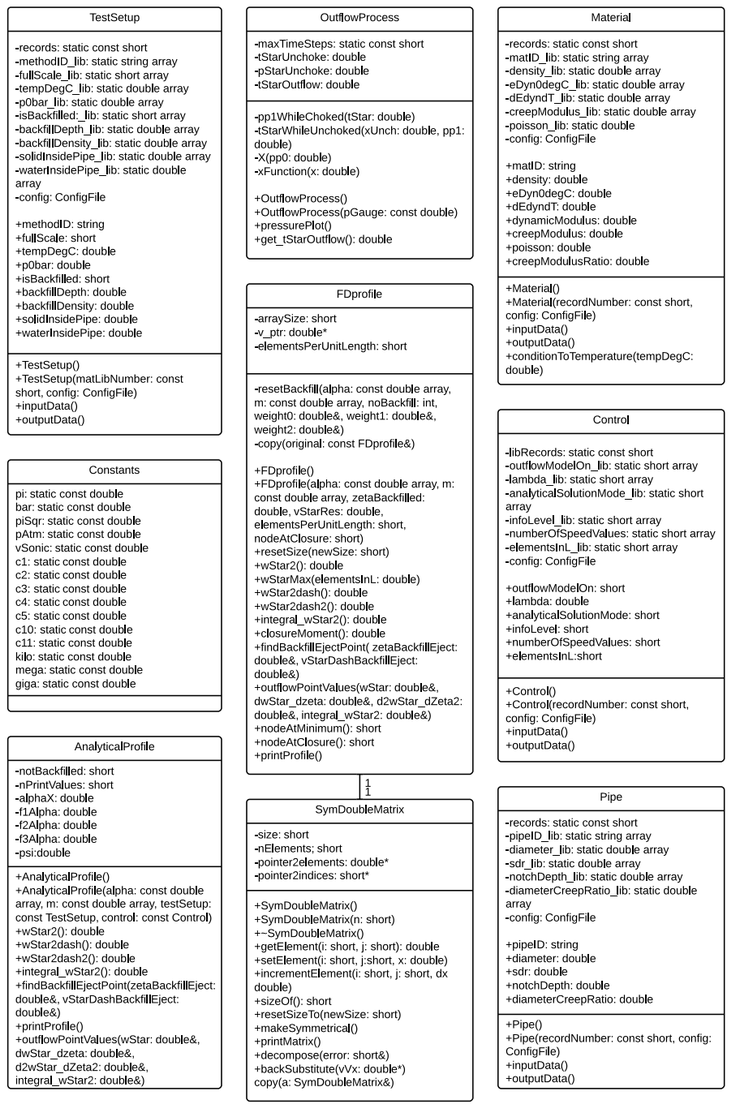


Figure.18: Original UML Map

A.2. UML: Final

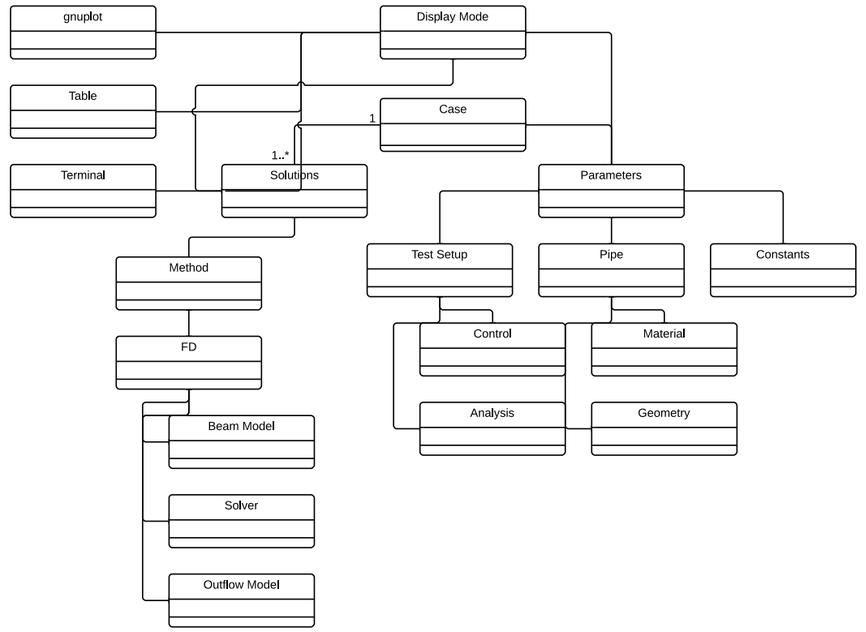


Figure.19: Final Terminal based UML Map

A.3. UML: GUI

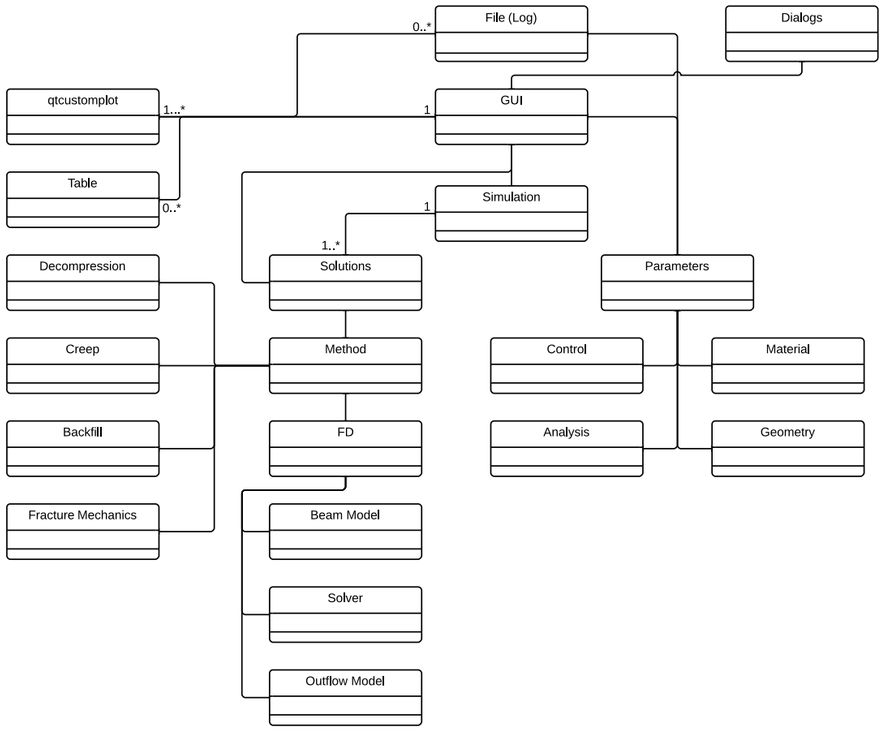


Figure.20: GUI UML Map

A.4. caseInputData.txt

Input Data

Material Data

matID = Generic PE80
density = 938
eDyn0degC = 2.62
dEdyndT = -0.037
creepModulus = 0.3
poisson = 0.38

Pipe Data

pipeID = 250mm_SDR11
diameter = 250
sdr = 11
notchDepth = 0.0
diameterCreepRatio = 1

Test Setup Data

fullScale = 2
tempDegC = 0.0
p0bar = 5.0
isBackfilled = 2
backfillDepth = 100
backfillDensity = 2200
solidInsidePipe = 0.25
waterInsidePipe = 0.25

Program Control Data

outflowModelOn = 0
lambda = 3
solutionmethod = 2
numberOfSpeedValues = 10
elementsInL = 10
aDotc0 = 0.5

A.5. MIT License

The MIT License (MIT)

Copyright (c) [2014] [Fraser Edwards][Dr Patrick Leevers]

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.6. Results.csv

Table.9: Results file

Normalised Crack Speed	Initial pressure	Temperature	Decomp. factor	Speed factor	Support factor	Outflow length	Flaring	Irwin Corten force	Crack driving force	Normalised force	CrackDarking	Converged	Iterations
0.02	5	0	0.144358	0.712399	3.90899	2.54788		0.689812	1.3673	0.810751	1	1	15
0.04	5	0	0.154128	0.818324	6.9049	3.01391		0.689812	1.57948	1.08953	1	1	12
0.06	5	0	0.164063	0.88209	8.4262	3.24251		0.689812	1.80962	1.24761	1	1	13
0.08	5	0	0.174256	0.934137	9.80394	3.40326		0.689812	2.04942	1.41330	1	1	13
0.1	5	0	0.186683	0.991713	11.3574	3.57587		0.689812	2.35564	1.62139	1	1	13
0.12	5	0	0.195378	1.04935	12.7879	3.71281		0.689812	2.68811	1.80419	1	1	13
0.14	5	0	0.206316	1.11749	17.3737	3.83323		0.689812	3.0658	2.12062	1	1	14
0.16	5	0	0.217513	1.21579	20.4139	4.27427		0.689812	3.5395	2.44035	1	1	14
0.18	5	0	0.228974	1.33432	22.643	4.19545		0.689812	4.33588	2.784	1	1	14
0.2	5	0	0.240753	1.41284	24.9908	4.3911		0.689812	4.98099	3.19031	1	1	14
0.22	5	0	0.252756	1.49827	26.826	4.40456		0.689812	5.20335	3.58033	1	1	14
0.24	5	0	0.264908	1.58291	28.302	4.49293		0.689812	5.87584	4.05330	1	1	14
0.26	5	0	0.277354	1.63787	30.3355	4.56742		0.689812	6.56302	4.53277	1	1	14
0.28	5	0	0.290411	1.71287	31.8904	4.82613		0.689812	7.37934	5.08706	1	1	13
0.3	5	0	0.303363	1.83111	33.7763	4.71881		0.689812	8.32981	5.75030	1	1	13
0.32	5	0	0.317816	1.94071	34.5902	4.7737		0.689812	9.2721	6.39651	1	1	13
0.34	5	0	0.330776	2.02978	35.5401	4.81300		0.689812	10.2584	7.0754	1	1	13
0.36	5	0	0.344909	2.12922	37.187	4.86378		0.689812	11.3752	7.84677	1	1	13
0.38	5	0	0.35824	2.23832	38.8237	4.93380		0.689812	12.4418	8.74809	1	1	13
0.4	5	0	0.373956	2.33434	40.2315	5.20052		0.689812	14.2348	9.81934	1	1	13
0.42	5	0	0.386003	2.39198	38.4494	4.93385		0.689812	14.756	10.1844	1	1	12
0.44	5	0	0.400387	2.47267	38.1128	4.82309		0.689812	15.8131	10.9095	1	1	12
0.46	5	0	0.420113	2.53206	37.8444	4.21431		0.689812	16.8351	11.6131	1	1	12
0.48	5	0	0.436189	2.63209	37.2533	4.39077		0.689812	17.7971	12.2757	1	1	12
0.5	5	0	0.452621	2.68251	34.9488	4.89945		0.689812	18.7478	12.9334	1	1	12
0.52	5	0	0.468416	2.73272	34.2139	4.83248		0.689812	19.4157	13.3932	1	1	12
0.54	5	0	0.48039	2.81736	34.6758	4.0602		0.689812	20.2019	14.0042	1	1	12
0.56	5	0	0.504119	2.96536	35.3836	4.85774		0.689812	20.6742	13.8676	1	1	11
0.58	5	0	0.512061	3.69886	40.548	5.55834		0.689812	11.0234	7.60608	1	1	22
0.6	5	0	0.540353	3.64949	61.4134	5.53171		0.689812	6.04618	4.17073	1	1	22
0.62	5	0	0.550061	3.71972	60.0872	3.48542		0.689812	3.84745	2.63492	1	1	13

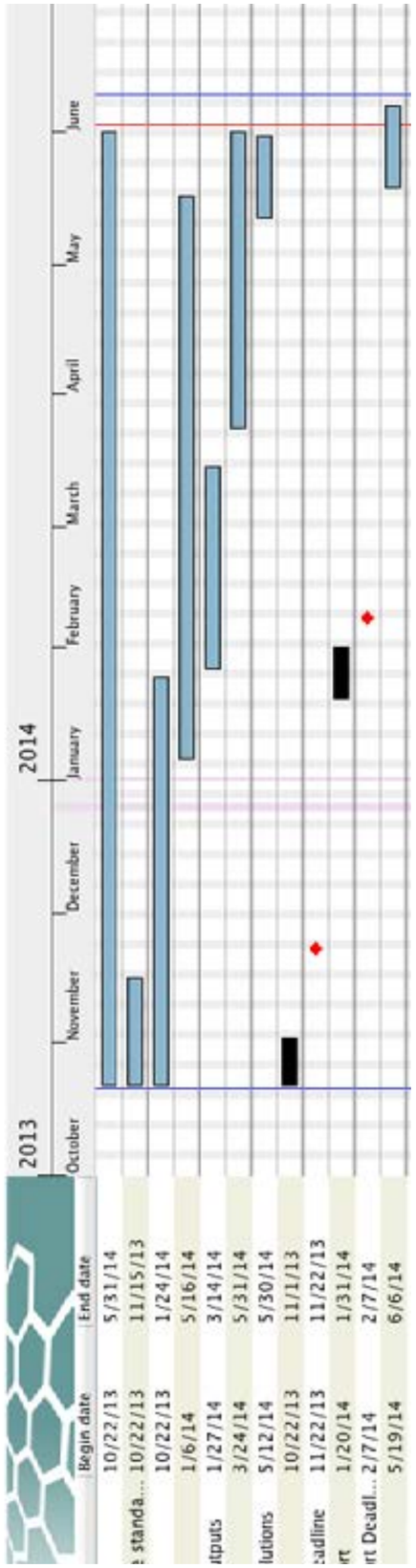
A.7 . Log.csv

Table.10: Log file

Program Control Data	
outflowModelOn	2
lambda	3
solutionmethod	2
numberOfSpeedValues	50
elementsInL	10
aDotc0	0.5

Log number	aDotc0	Irwin Corten Force	diameterRes0	residualCrackClosure	densityratio	zetaclosure	nodeAtClosure	outflowLength	pbar	v0	vStarRes	factor	aDotCLfactor	aDotCifactor_backfilled	maxIterations	iterations	m[0]	m[1]	alpha[0]	alpha[1]	error	notConverg
1	0	0	0	0	12.589	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	12.589	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
3	0	0.689812	0	0	12.589	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
4	0	0.689812	250	0	12.589	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
5	0.02	0.689812	250	0	12.589	2	20	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	0	0	0	0	0	0	0	1
6	0.02	0.689812	250	0	12.589	2	20	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	0	6.60879	6.60998	0.838365	0.812568	0	1
7	0.02	0.689812	250	0	12.589	2	23	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	1	6.60879	6.60998	0.838365	0.812568	0.196326	1
8	0.02	0.689812	250	0	12.589	2	24	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	2	6.60879	6.60998	0.838365	0.812568	0.171341	1
9	0.02	0.689812	250	0	12.589	2	25	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	3	6.60879	6.60998	0.838365	0.812568	0.147753	1
10	0.02	0.689812	250	0	12.589	2	26	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	4	6.60879	6.60998	0.838365	0.812568	0.125645	1
11	0.02	0.689812	250	0	12.589	2	27	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	5	6.60879	6.60998	0.838365	0.812568	0.1051	1
12	0.02	0.689812	250	0	12.589	2	28	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	6	6.60879	6.60998	0.838365	0.812568	0.086198	1
13	0.02	0.689812	250	0	12.589	2	29	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	7	6.60879	6.60998	0.838365	0.812568	0.0689968	1
14	0.02	0.689812	250	0	12.589	2	30	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	8	6.60879	6.60998	0.838365	0.812568	0.0535296	1
15	0.02	0.689812	250	0	12.589	2	31	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	9	6.60879	6.60998	0.838365	0.812568	0.0398003	1
16	0.02	0.689812	250	0	12.589	2	32	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	10	6.60879	6.60998	0.838365	0.812568	0.0277826	1
17	0.02	0.689812	250	0	12.589	2	33	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	11	6.60879	6.60998	0.838365	0.812568	0.0174222	1
18	0.02	0.689812	250	0	12.589	2	34	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	11	6.60879	6.60998	0.838365	0.812568	0.0086403	1
19	0.02	0.689812	250	0	12.589	2	34	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	11	6.60879	6.60998	0.838365	0.812568	0.0086403	1
20	0.02	0.689812	250	0	12.589	2	34	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	11	6.60879	6.60998	0.838365	0.812568	0.0086403	1
21	0.02	0.689812	250	0	12.589	2	34	3	0.721942	0.175565	0	1.33413	1.00002	1.0002	50	11	6.60879	6.60998	0.838365	0.812568	0.0086403	1
22	0.02	0.689812	250	0	12.589	2	34	2.63092	0.721942	0.103844	0	1.33413	1.00002	1.0002	50	12	6.60879	6.60998	0.838365	0.812568	0.0086403	1
23	0.02	0.689812	250	0	12.589	2	34	2.63092	0.721942	0.103844	0	1.33413	1.00002	1.0002	50	12	3.90899	3.9097	0.735222	0.712599	0.0086403	1
24	0.02	0.689812	250	0	12.589	2	37	2.63092	0.721942	0.103844	0	1.33413	1.00002	1.0002	50	13	3.90899	3.9097	0.735222	0.712599	0.0172215	1
25	0.02	0.689812	250	0	12.589	2	38	2.63092	0.721942	0.103844	0	1.33413	1.00002	1.0002	50	14	3.90899	3.9097	0.735222	0.712599	0.0101326	1
26	0.02	0.689812	250	0	12.589	2	39	2.63092	0.721942	0.103844	0	1.33413	1.00002	1.0002	50	14	3.90899	3.9097	0.735222	0.712599	0.00406483	1
27	0.02	0.689812	250	0	12.589	2	39	2.63092	0.721942	0.103844	0	1.33413	1.00002	1.0002	50	14	3.90899	3.9097	0.735222	0.712599	0.00406483	1
28	0.02	0.689812	250	0	12.589	2	39	2.63092	0.721942	0.103844	0	1.33413	1.00002	1.0002	50	14	3.90899	3.9097	0.735222	0.712599	0.00406483	1
29	0.02	0.689812	250	0	12.589	2	39	2.63092	0.721942	0.103844	0	1.33413	1.00002	1.0002	50	14	3.90899	3.9097	0.735222	0.712599	0.00406483	1

Appendix



A.9.1. Github

Github has worked fantastically throughout the project, aborting many disasters by maintaining a working version of code at all times. Furthermore, it allowed major changes to be made without having to worry about ruining the underlying model.

There are a few drawbacks to using the Github application as opposed to using it through the CLI such as the inability to create releases.

A.9.2. gnuplot

Extremely easy to learn and implement, gnuplot provides incredible functionality without an in-depth knowledge of the source code required. However, while gnuplot is easy to operate through terminal, controlling it from a program was slightly more difficult. A number of C++ libraries to handle this were available, which functioned perfectly well but weren't very well documented. The available functions were discovered by scouring the header and example source file.

Unfortunately, despite being the most complete in terms of functions did not have one for closing temporary files. This along with operating system limits restricted the number of plots which could be produced in any simulation and hence the number of speed iterations.

Furthermore, gnuplot requires installation. Without this installation, aRCPlan's outputs are reduced to tables and text displayed through Terminal. This installation also adds a step before aRCPlan can be used to it's full potential. A caveat to this is that the majority of researchers are likely to have a version of gnuplot installed due to it's extensive use.

A.9.3. qtcustomplot

qtcustomplot is a custom widget utilising Qt's QCPPainter class to create graphs. It is extremely well documented for a single developer project with a forum maintained and regular updates. The implementation of this plotting service allows it to be dragged and dropped in to the Qt Creator form designer making it extremely quick to realise.

It provides much the same functionality as gnuplot with options to customise the line style, thickness and colour as well as more in depth options. There are two main advantages to qtcustomplot over gnuplot: no need for installation and no limit to the number of plots produced. However, as the name suggests, qtcustomplot will not function outside of qt applications.

A.9.4. Sublime

A text editor was chosen for it's simplicity as opposed to Xcode, described in §1.2.3. Whilst a range of text editors were trialled, Sublime became the preferred choice. With capabilities such as split view and a sidebar showing the current view location in the overall source code, Sublime provides a basic but well formatted environment to edit code in.

At the time of writing, Sublime is available free for a trial period, with continued use requiring a license, available for purchase online.

A.9.5. Qt

Qt was chosen over Xcode and wxWidgets for the development of the GUI mostly for it's intuitive interface and modular structure, meaning the program scales depending on the number of platforms which will be developed for. Furthermore, Qt is not platform specific, unlike Xcode. As a result, if the project is compiled in Qt running in Windows, the resultant

executable is produced with a native style.

A downside of this is that the for clean environments, systems without Qt installed, the build size can become substation. The original program running in Terminal was 1.1 MB, increasing to 23.3 MB once the GUI was introduced.

A.9.6. Xcode

Although powerful, Xcode was rejected for being too powerful and bulky to allow efficient development. Furthermore, any programs it produces are platform specific; it can not produce executables for Windows.

Although it was unused, its installation was required to access the command line tools required for development. If development is continued in Qt, Xcode may not be needed.

A.9.7. Materials (Courses)

At present, only the Embedded C for micro controllers course provides materials on coding outside of the computing courses which deal only with the Matlab software. Moreover, the course only deals with C and concentrates on use in a microprocessor. As a result, the main advantages of C++ were learnt through research and finding solutions to problems. Furthermore, external skills such as writing makefiles and creating aliases were learnt in an ad-hoc manner. A course on programming focusing on computer science rather than higher level languages such as that used in Matlab would have been extremely useful.

Materials (External)

A consequence of the lack of dedicated course to C++ was a heavy reliance on outside sources such as Stack Overflow, a forum dedicated to programming. Stack Overflow was used heavily to find solutions to various problems throughout the project. As a general rule, any problem experienced during the project already had a solution on the Stack Overflow website. The Qt forum provided a similar function with suggestions for the majority of problems experienced.