Programming for Longer Battery Life

Anna Frances Rasburn

**April 2018**

**Dissertation submitted in partial fulfilment for the degree of   
Bachelor of Science with Honours in Computing Sciences**

**Division of Computing Science and Mathematics  
University of Stirling**

Abstract

Problem

The surge in the dependence on mobile phones and portable require consideration on the environment this dependence has had. This dependence is due to the fast advancement in mobile technology and the convenient nature of mobiles but has also led to users wishing for a mobile battery to last longer. Mobile applications are a significant drain on battery due to the higher demand on the device's CPU. This project's aim is to research and investigate the ways that android java applications can be modified to reduce CPU power and therefore require less battery.

This project focuses on using optimisation techniques applied to java particularly using genetic improvement and small changes that developers that can make within their source code that could have a significant change in the requirements of CPU power.

Methodology

This project had a creation of a software application which applies edits to source code. The edits created were finding the impact of: moving, deleting, swapping block statements, other edits included inserting lines and changing the use of If statements. The application uses artificial techniques such as Hill Climbing and Genetic Algorithms to find the most optimised version of the source code.

Achievements

Attestation

I understand the nature of plagiarism, and I am aware of the University’s policy on this.

I certify that this dissertation reports original work by me during my University project except for the following (*adjust according to the circumstances*):

* The technology review in Section 2.5 was largely taken from [17].
* The code discussed in Section 3.1 was created by Acme Corporation ([*www.acme-corp.com*](http://www.acme-corp.com)*/JavaExpert*) and was used in accordance with the licence supplied.
* The code discussed in Section 3.5 was written by my supervisor.
* The code discussed in Section 4.2 was developed by me during a vacation placement with the collaborating company. In addition, this used ideas I had already developed in my own time.

**Signature** *(you must delete this, then sign and date this page)* **Date**

Acknowledgements

Acknowledge anyone who has helped you in your work such as your supervisor, technical support staff, fellow students or external organisations. Acknowledge the source of any work that is not your own.

Table of Contents

The table of contents below is automatically generated from the paragraphs of style *Heading N* and *Unnumbered N*. To update this after revisions, right-click in the table and choose *Update Field* for the entire table.

Abstract i

Attestation ii

Acknowledgements iii

Table of Contents iv

List of Figures v

1 Introduction 1

1.1 Background and Context 1

1.2 Scope and Objectives 1

1.3 Achievements 1

1.4 Overview of Dissertation 1

2 State-of-The-Art 2

3 Technical Chapters (change this to something appropriate) 3

3.1 First Section 3

3.1.1 First Subsection 3

3.1.1.1 First Subsubsection 3

3.1.2 Second Subsection 3

3.2 Second Section 3

4 Conclusion 5

4.1 Summary 5

4.2 Evaluation 5

4.3 Future Work 5

References 6

Appendix 1 7

Appendix 2 – User guide 8

Appendix 3 – Installation guide 9

List of Figures

Similarly, you can automatically generate a list of figures from paragraphs of style *Figure*. To update this after revisions, right-click in the table and choose *Update Field* for the entire table.

[Figure 1. Highly Technical Diagram 18](#_Toc534200613)

# Introduction

For editorial consistency, it is important to use Word styles properly. Word 2003 onwards has so-called quick styles. If the styles referred to below are not visible on the *Home* ribbon in the *Styles* category, choose *Apply Styles* from the down arrow at the bottom right of the *Styles* category. Styles can then be applied from the drop-down box. To make a style visible as a quick style, choose *Apply Styles*, then click *Styles* (the *AA* icon), then click on the drop-down list for a style, and then *Add to Quick Style Gallery*.

Chapters are entered using the Heading 1 paragraph style. The Heading 1 style automatically moves to the start of a new page and supplies the next chapter number. The new paragraph when you press Return after a heading automatically uses the *Body First* paragraph style (like this one, with no indent on the first line).

However, the most text uses the Body Text paragraph style (like this one, with 11 point Times New Roman, 1.5 line spacing, single-sided pages). Enter most text using the Body Text paragraph style. The new paragraph when you press Return after a Body First paragraph automatically uses the Body Text paragraph style.

In general, use the default spacing that headings and paragraphs give you. Avoid using new-lines or spaces to format text. If you need to use quotes, preferably use single curly quotes ‘…’. If you wish to emphasise something, usually use *italic font*.

**Remember to Save frequently while you are working!**

## Background and Context

Give the background to your project and context of what you have done. Sections are entered using the *Heading 2* paragraph style – th*e Heading 2* style automatically supplies the next section number.

A large proportion of this research is building on research previously conducted by Dr Alexander Brownlee [REFERENCE] and Dr David White [REFERENCE] which focuses on methods used to optimise source code without losing any functionality through Genetic improvement. This research also requires the use of previously undertaken research by Bruce et al [REFERENCE], which considered methods to automatically change source code in order to reduce energy usage.

### Optimisation regarding Java

Optimisation is ‘the act of making the best or most effective use of a situation or resource'. Optimisation in Java focuses on goals within the source code that we can optimise [REFERENCE]. This research focuses on changing Java code and Projects that can be used to improve an applications’ execution time and reduce an application’s requirement of energy needed to run.

In order to find solutions to optimisation problems, a technique is Genetic Algorithms SEE BELOW which find solutions using the notion of fitness SEE BELOW.

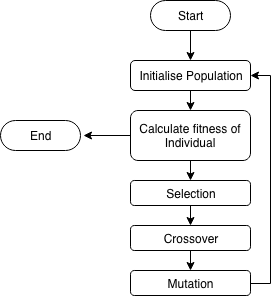
### Genetic Improvement

Genetic Improvement(GI) improves software by using machine learning and optimisation techniques. These techniques include Evolutionary Algorithms and therefore Genetic Algorithms. Genetic Improvement can be used to improve pre-existing software which improvements such as reducing energy consumption, execution time and memory consumption.

### Genetic Algorithms

Genetic Algorithms(GA) are algorithms based on the biological notion of evolution and natural selection as devised by Charles Darwin. Genetic algorithms use the survival of the fittest and inheritance so fittest traits of the parent are passed onto the children. Throughout the duration of the algorithm, many possible solutions to a problem are found, these solutions are called ‘individuals'. Every iteration of the algorithm finds a solution with the fittest traits in order for improved solutions each time and traits that are not desired are disregarded and not passed on.

In standard Genetic Algorithms, there are five main steps that are followed, this is seen in FIGURE \_. The steps are initialising population, fitness, selection, crossover and mutation.



**Figure 1** Flow Diagram of a Genetic Algorithm

An “Individual” or “chromosome” is a possible solutions to the problem. In a classic GA, a chromosome in a string of bits and each individual bit is a gene. The selection stage ‘selects’ individuals with high fitness that will progress to the next population. In the mutation stage, individual genes of the highly fit parent chromosome are flipped from one to zero or vice version in order to create a new mutated chromosome to pass onto further populations. The crossover stage requires taking genes from the two parents and combine them to make a new individual who should be a more optimised solution to the problem than their parents.

A “Population” is a set of individuals. The initialise population state is gathering a collection of individuals together to form a population. As individuals increase their fitness, the overall population has a higher fitness and if therefore has a higher possibility of finding a solution to the problem.

In other problems such as Genetic improvement problems, the individuals can be lists of changes in source code or new versions of the source code. In the GI tool Gin [REFERENCE], an individual is a list of "Edits". Each "Edit" is a single change to the source code such as moving a line from one place to another. The mutation stage adds or removes edits to source code and in the crossover stage, a combination of edits are combined together into a new version of the source code. An edit is similar to a gene but not a direct parallel because an edit is what has changed in the code. A "Patch" is similar to a chromosome but not the traditional definition as a "Patch" may compromise of no edits in case the most optimised version of source code in the most optimised source code. A patch could also contain a collection of edits to create that optimised source code. In the Gin Code[REFERENCE], a collection of patches is similar to a classic Genetic algorithm’s definition of a population.

[REFERENCE THIS- USE THAT GA BOOK]

### Fitness

As mentioned previously, the fitness of an individual is how well that particular individual performs the task. In this instance, there are several potential measure of fitness: does the new source code compile? does the new source code pass the unit tests? And the energy consumption of the application. This has meant that each element of fitness needs a weight which is how important that element is in the fitness function. For example, whether the source code can compile has a larger weight than the unit tests because if a program does not compile then the unit tests are not important as the program has lost all functionality. Table 1 presents the different elements of the fitness and a way that the different weights can be used together, in the below example individual 3 would be the most optimised because it has passed the most weighted elements such as it can compile and energy consumption although it failed two unit tests.

**Table 1** Example to Demonstrate the levels of Fitness

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Did the source code compile?** | **Unit tests** | | | **Energy**  **Consumption** | **Use this**  **individual?** |
| **Unit test 1** | **Unit test 2** | **Unit test 3** |
| *Individual 1* | Yes | P | P | P | 3000J | Y |
| *Individual 2* | N | P | P | F | N/A | N |
| *Individual 3* | Y | P | F | F | 2100J | Y |
| *Individual 4* | Y | F | P | P | 4000J | Y |

For any optimisation problem, the fitness function needs to be defined before a solution is executed. When you change the fitness function you can find new ways to optimise Java. Changing the fitness to focus on execution time or energy consumption, the Java application can be optimised in different ways.

## [Scope](http://www.cs.stir.ac.uk/~kjt/research/conformed.html) and Objectives

Define the scope and objectives of your project.

### Scope

The scope of this research is to discover different ways developers can change their source code in order to make more optimised applications that firstly have a shorter execution time but also achieves the same functionality. This research will be followed by looking at the same principles and seeing whether we can optimise applications to require less energy. Furthermore, I hope this research will lead to the development of a toolkit that uses elements of artificial intelligence methods such as Genetic Algorithms and local search that uses a developers source code and well-written unit tests to create a more optimised version of the original source code.

### Users

The users of this research would be android application developers and Android application users. The Android mobile users wish to have a device which does not ‘die' as often and this device uses applications that are unlikely to ‘drain’ the battery. This research can also apply to more general Java developers as this research should be applicable to larger applications and devices such as a computer where the lower energy consumption will have a greater impact on the environment.

### Objectives

The objective for the project is:

1. To research the different ways we can optimise Java, specifically looking at changing operators in the code to see its effect and the use of Genetic Improvement to optimise Java. What kind of changes can be made to Java code in general?
2. Applying the research from earlier to create some software in a similar vein to the Gin code [REFERENCE] that uses genetic algorithms to change the source code, so it is less execution and energy intensive.
3. To learn about and use the existing Opacitor [REFERENCE] and Jalen tools [REFERENCE] to approximate the energy consumption of the changes Java code and prove that the experiments have been successful.
4. To acquire an existing piece of software and applying GI techniques so it has a shorter execution time.
5. To acquire an existing piece of software and applying GI techniques so it requires less energy to run.

## Achievements

Summarise what you have achieved.

Toolkit

I began the project by creating a toolkit in a similar vein to the GIN code base [REFERENCE]. The toolkit had additional functionality of being able to take an entire project at once.

I used the GIN code as an inspiration.

The toolkit also has the test runner and Opacitor running in one movement. The Gin code had no Opacitor connecting.

Genetic Algorithm

## Overview of Dissertation

Briefly overview the contents of what follows in the dissertation.

# State-of-The-Art

Summarise current knowledge and what others have done in the various topics of your dissertation – in the application area and in the various technologies that you might have used or did use. Write for someone familiar with computing, but not necessarily expert in the particular topics of your project. Give references to other work by using *cross-references* to entries in the References section, like this [2] (see the References tab in Microsoft Word's ribbon).

It is important to write a *critical* literature review that identifies gaps in current solutions and that clearly shows how the project was driven to address these gaps. This chapter should, therefore, feed into well-defined requirements for the project. Avoid a superficial description of related work that does not carefully analyse its strengths and weaknesses.

State of art split into 3 sections:

Genetic Improvement/Changing Software

Energy Consumption

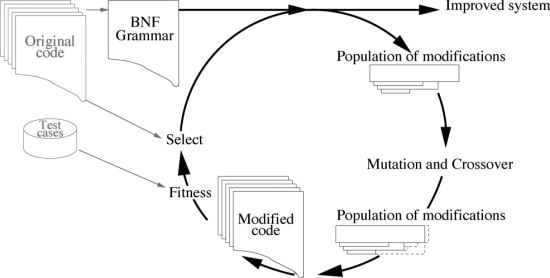
Green Software Engineering

## Genetic Improvement

Genetic Improvement is used to improve pre-existing software by using an automated search to find improved software. These improvements include improving energy consumption, reducing execution time and memory consumption. In a survey of 3132 distinct papers that were published between 1995 and 2015 [REFERENCE - Genetic Improvement of Software: A Comprehensive Survey], the most popular search technique is evolutionary computing as 96% of these papers referenced or used these techniques, this demonstrates evolutionary computing increase in popularity.

This survey is advantageous as it is one of the largest and comprehensive surveys on computer science papers and conclusively found that there were 82 papers out of 1131 that was about genetic improvement [REFERENCE].

However, in order to cover a wide range of topics that fall within the definition of genetic improvement, the analysis for the survey had to be restricted. This may have led to many papers being excluded and not included within the survey which may have key information about genetic improvement but are excluded due to the survey's strict definition.

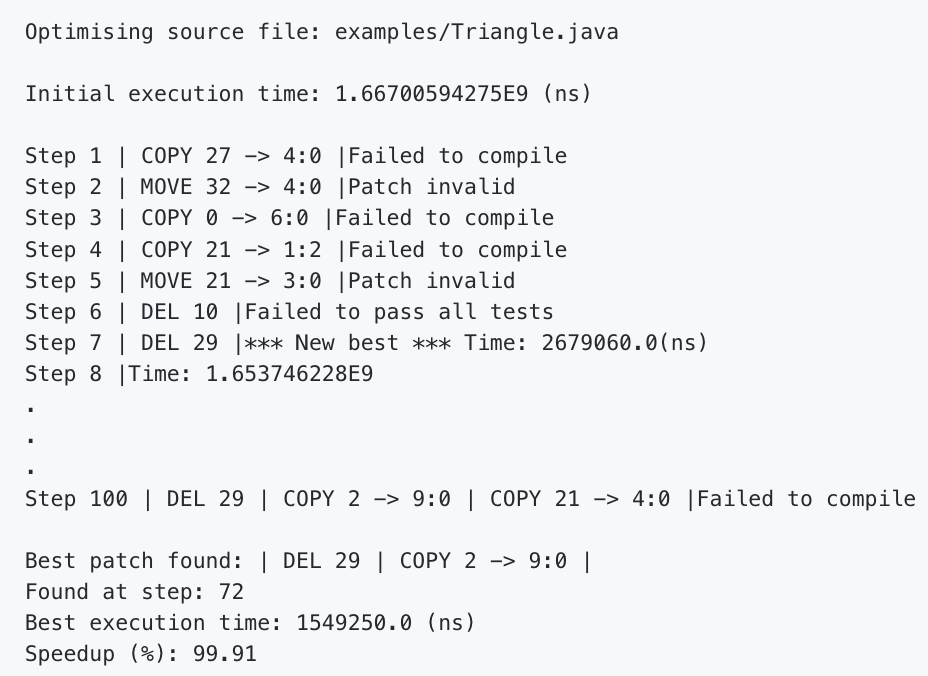


**Figure 2** Diagram of changing code lifecycle with Genetic Improvement

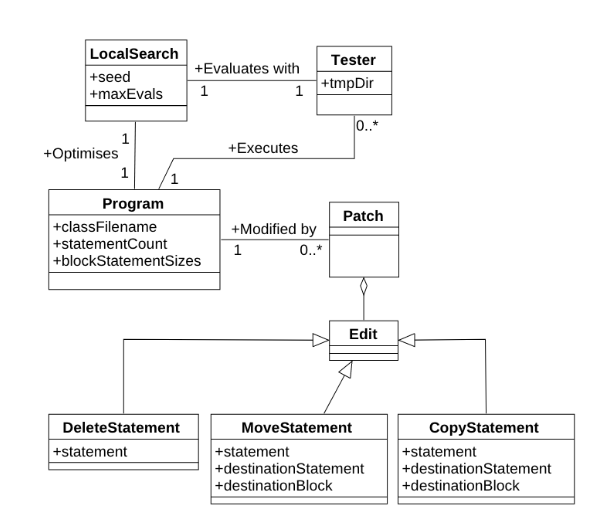
A lifecycle of improving software using genetic improvement can be seen in Figure . Langdon and Harman [REFERENCE] state that the ‘original code is a state oracle’, therefore giving a programmer a benchmark to compare and reassure that it is the expected output. Genetic Improvement tries to keep the same functionality as the original code which can lead to improving the outputs generated as well as optimising the code. However, this research [REFERENCE] assume that the original code is correct and is passing unit tests. Therefore, this ignores integration testing and only focuses on genetic improvement for individual sections of code rather than considering a complete system.

## GIN

GIN, Genetic Improvement in no time, is a Java tool that makes it easier to implement Genetic Improvement and has a goal to “to stimulate development in GI tooling”[REFERENCE – GITHUB][REFERENCE-PAPER]. The Gin application itself optimises a Java class, using a local search to add “Edits” to the source code till a more optimised version of the code is found. As seen in Figure 2 and 3, GIN manipulates code in three major ways: deletion, copying and replacement. For instance, particular lines of code or blocks of code could be deleted such as in the ‘Triangle.java’, the removal of the reference to the ‘delay()’ method creates a more optimised version of the code that passes all the unit tests and has a lower execution time. An example of the Gin code output is seen below.



**Figure 3** Example of GIN output



**Figure 4** UML diagram of how GIN works

The GIN toolkit is experimental and allows for the development of skills and knowledge about optimised code and GI techniques that could later be integrated into the JVM and compiler.

The GIN toolkit is one of the very few tools looking at integrating GI within a Java project. It applies a variety of edits to source code and concludes in the most optimised version of the source code. The toolkit does create optimised source code with shorter execution time, its use of edits will be seemed as ‘small changes' to the developer but are shown to have a large impact on execution time and optimisation.

The GIN toolkit uses a local search to make decisions on how to apply edits to the source code. The use of a local search allows the ability to find reasonable solutions in continuous state space so, in the gin example, there are many neighbouring solutions made quickly that do lead to an optimised version of the Triangle example source code. Figure 3 presents a UML diagram of how UML works, as seen, the Local Search class is used to evaluate the tests and to optimise the program.

The fittest function of the GIN toolkit is based on whether the Junit tests are successful as it is the thought that the edited source code must have the exact same purpose as the original source code as the unit tests are still passed. However, this fittest function implies that the developer has created well-written unit tests and source code. Therefore finding an optimised version depends largely on the developers' initial code.

The GIN toolkit is limited to only optimise Java code. One can imagine a lot of the principles that are learnt from this software could be applied to other object-oriented languages like C. Some of the features such as moving edits can have a major impact on interpreted languages such as python which have every line of the script interpreted individually where the position of lines are important when optimising source code.

## JCodec

JCodec is ‘a pure implementation of video and audio codecs’[REFERENCE]. JCodec is currently being used by the GIN toolkit as a set of programs to improve. There are 203 different programs within JCodec. JCodec is a good library for GIN to optimise because they are written in Java and they contain more than 10 unit tests per program.

The high number of unit tests allow for the functionality not to be comprised largely so the programs still perform in the same way. The JCodec implementation is used in a large number of programs and large numbers of developers. This created a lot of documentation and if GIN finds a way of improving the library, there can be large saving in energy if all of those programs use the optimised versions.

JCodec offers its own problems when optimising because a lot of the programs use media elements which require a lot of energy to work and cannot have too much content removed otherwise you remove the functionality of the program and cause a wide range of errors. The research on this library does not work on other multimedia techniques in Java such as the Oracle multimedia API [REFERENCE]as they work in different ways in comparison to JCodec.

<http://jcodec.org/>

<https://www.oracle.com/technetwork/database/database-technologies/multimedia/documentation/index.html>

## GI In other Languages

<https://dspace.stir.ac.uk/bitstream/1893/26007/1/thesis.pdf>

<https://www.economist.com/science-and-technology/2018/07/19/python-has-brought-computer-programming-to-a-vast-new-audience>

A large proportion of research about Genetic Improvement focuses on object-oriented languages such as Java, C and C++, therefore ignoring dynamically typed languages such as Python. Research conducted by Haraldsson[REFERENCE] focuses on genetic improvement in Python can be used to fix multiple bugs within the Python script at the same time and in a reasonable time frame.

Haraldsson's research works in a similar way to GIN where a series of edits are applied to source code. This research applies experiments to small python scripts to see whether solutions created by genetic improvement has been successful. The use of a Random walk analysis where the start state is assuming the program is correct and the fitness is monitored as the edit list increases in size one edit at a time. The other stage of the experiment is the exhaustive neighbourhood analysis were the fitness is evaluated for every mutant of the program.

Genetic Improvement in Python is beneficial because the search space is less restricted than the search space for Java programs. This allows for more changes to the source code in a Python Script in comparison to small Java programs. Optimising Python is important because Python is ‘becoming the world’s most popular coding language’[REFERENCE] therefore the increasing number of python developers can code an optimised version of the language with bugs fixed quickly in a reasonable time.

The larger search space of optimising Python requires a longer time to achieve results due to fewer restrictions. The results produced are also difficult to verify and prove that the optimised script has the same functionality as the original script as there are no unit tests which is used a measure of fitness in Java and OOP languages. This research is valuable as a lot of the concepts about genetic improvement can be used in my research but Haraldsson's research does not fully apply as it focuses on Python rather than Java.

## TODO: Another GI example – Look at paper references (Maybe)

<https://link.springer.com/chapter/10.1007/978-3-319-22183-0_23>

<https://link.springer.com/chapter/10.1007/978-3-319-22183-0_20>

## Opacitor

Opacitor [REFERENCE] is a tool which measures the energy consumption of JVM programs. It does this by using a bytecode level model of energy cost. Opacitor ‘can detect small changes in execution profile, down to opcode level’ [REFERENCE] therefore it records the energy used by a program down to its lowest level. The Opacitor measures energy usage in joules and does not use a CPU as a proxy for energy use like lots of other similar tools have. Therefore, using joules and regarding bytecode, increases the accuracy of the readings.

Opacitor [REFERNCE][REFERENCE] is one of the first tools of its kind to measure the energy produced of an executing program using Search-Based Software Engineering, Due to the implementation of the Opacitor it can detect the smallest changes in execution, therefore recording all the possible energy usage from the entire execution of the program. The Opacitor is unaffected by the rest of the computational environment so when it examines the smallest changes these are the only changes in the program and does not include any other environmental factors.

The Opacitor is deterministic therefore it will produce the same output from a given input. The Opacitor’s results have helped reduce a program’s energy by up to 70% in the best case and 20% in the worst case [REFERENCE]. The Opacitor is one of the few tools that can detect small changes in the execution and responsive to metaheuristic searches[REFERENCE].

The Opacitor's only focus is on the CPU and does not consider other features of a mobile phone that uses a large amount of battery power such as the display, WI-FI and GPS. The Opacitor is exclusive to applications and the CPU does not consider of the other features or how they work with applications and their effect that they may have on the battery.

## Jalen

<https://tel.archives-ouvertes.fr/tel-00961346v2/document>

Jalen is a ‘runtime measurement software for estimating the energy consumption at code level for Java applications’ [REFERENCE]. Jalen uses information provided through application-level monitoring tools to calculate the energy used by an application. The energy information is estimations from the levels of threads and methods, the information collected is method durations, CPU time and so on. The information gathered is used to make statistics which is used to calculate the energy consumption as well as using energy models seen in Figure 5. The energy consumption is calculated per method and exposed as a service. Jalen can provide information about Net Energy, Net Library energy and all energy which includes energy consumption of all methods running in the JVM.



**Figure 5** Architecture of Jalen

Jalen offers a valuable tool that provides an energy count on a code level so there is an increased understanding at the energy consumption and distribution in software. Jalen is also one of the very few toolkits that detect energy hotspots in code and knows what particular areas of software are energy intensive. Jalen has been done with experiments on a range of different program from simple algorithms to real-world scenarios showing its versatility and the potential of energy code profiling. The information gathered from Jalen will help developers find out which methods were energy intensive so they can change the ways they code to ‘reduce the energy footprint of their application' [REFERENCE].

Unlike Opacitor, Jalen prefers statistical sampling over bytecode instrumentation. It also refers to providing percentages rather than raw Joules values. This is because that changing hardware changes the results received which is not a problem that Opacitor finds. Offering the results as a percentage means that results do not change drastically when the hardware changes like it do when it is a Joule's count.

## Optimization Framework for Mobile Applications.

<https://arxiv.org/pdf/1608.05248.pdf>

<https://ieeexplore.ieee.org/document/7935484/references#references>

Li et al created a framework which looks at Source-level energy optimization with a particular mobile application [REFERNCE]. This framework proposed an energy-optimization framework with a source code energy model that allows developers to know about energy usage. This framework changes and improves the code to save CPU energy by analysing energy features of the source code before optimisation. The model maps the energy use to the basic operation within the source code which works in a similar way to Jalen as it focuses on the energy impact on individual operations within the source code.

This framework was a success as it was able to save between 6.4% to 50.2% of the CPU energy in various mobile applications. These toolkits and frameworks highlight to developers which operators are energy intensive so they may make better choices when developing an application. This framework focuses on mobile applications which are different to Jalen, this is valuable because the way mobile applications interact with mobile hardware is very different from the way software interacts with other machines.

Unlike the Opacitor, this framework does not look at the bytecode level, therefore, may have its results affected by features of a mobile device such as WI-FI and GPS. These features would have a major impact on the results because they are energy intensive and will increase energy usage results massively. The reliance of highlighting to a developer their energy usage means that for there to be any save in energy, this is entirely the developer's responsibility and then trusting them to be energy conscious.

## TODO: Code Refactoring’s effects on Energy Usage - how code changes affect energy consumption and which kinds of changes are likely to have the greatest impact

<https://www.researchgate.net/publication/266661686_How_Do_Code_Refactorings_Affect_Energy_Usage>

Research conducted by Sahin et al [REFERENCE – code refactoring] looked at energy impacts of 197 applications when 6 commonly-used refactoring techniques have been applied to those applications and the change in energy usage those applications have after those techniques applied. The results of this research concluded that there was an impact on energy usage and a change in refactoring techniques can effectively increase and decrease the amount of energy used by an application. This study conducted 394 experiments on those 197 applications and found that 28% (109 experiments) indicated ‘a statistically significant difference in energy usage between the original and refactored versions.’[REFERENCE]. The refactoring technique of ‘Convert Local Variable to Field’[REFERENCE] had the largest positive impact on energy out of the six different techniques. This study is useful in that it presents the effects of different refactoring techniques and has a difference in results. However, this study could have been extended to include more applications and refactoring techniques as it makes it harder to generalise. This study uses only two JVMs: 6 and 7, which does not consider whether new versions of Java have integrated optimisation techniques that affect energy usage.

This research supported by research conducted by Morales et al [REFERENCE – Anti-patterns: <https://arxiv.org/pdf/1610.05711.pdf>] who focuses on the effects of antipatterns in mobile applications and its effect of energy efficiency. This research found that energy consumption of apps with anti-patterns are not statistically different to apps who do not have these anti-patterns. The research did conclude that refactoring anti-patterns can be positive or negative therefore developers need to consider the energy impact of refactoring when creating applications. The research was conducted with 59 android applications and showed that applications containing anti-patterns consume more energy than those which do not have them, therefore refactoring and removing these antipatterns has a major effect on energy usage. Similar to the Sahin's study, this study does not use a lot of programs, therefore, making it harder to generalise. This study is very focused on Android applications and considers elements of applications that only affect Android.

## Green Software Engineering

Green Software engineering, a term coined by Hindle [REFERENCE] is a type of software engineering which is more environmentally sustainable so it takes into consideration the power usage of creating software. There are very few uses of physical measurements in software engineering which is critical for measuring energy which creates limitations regarding when it comes to experimentation and analysing energy usage.

Green Software Engineering is an important yet young field. This age, therefore, has led to a lack of resources such as a small coherent community and not agreed upon methodology. This field needs an increase in support and research which can pose a negative image of making the field an ‘impossible bar' [REFERENCE] that may not be achieved. An increase in support and research may come with the UN's '12 year limit' on climate change [REFERENCE]. This issue has only aplimifed the importance and necessity of this research if the desire to keep our current technology-heavy lifestyles as well as to ‘limit climate change'.

Research conducted by Hindle as it focuses on software’s interaction in other contexts other than mobile applications. Hindle researches in mobile applications, Data Centres and embedded sensors. Hindle has a proposal to have an energy rating on mobile applications included with the mobile application other functions, an example of this can be seen in Figure 6. Therefore, the user has the responsibility of choosing a more sustainable application which offers an interesting proposal as it gives the user more responsibility and can be used an extra selling point for a mobile application.

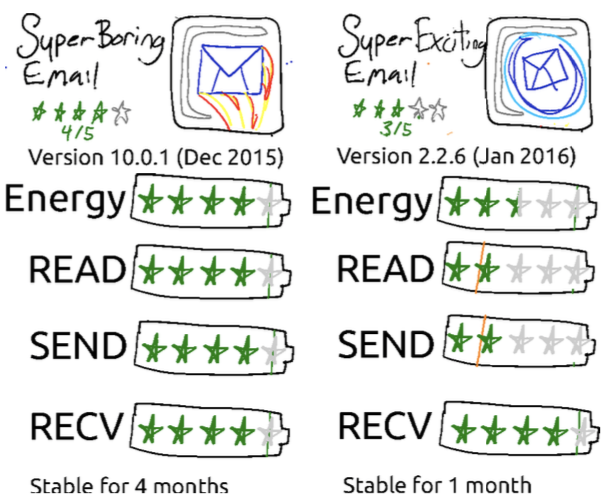


Figure 6 Ratings for Mobile applications

All of the previous research mentioned is useful in regard to green software engineering. As the combination of changing source code using Genetic Improvement to reduce energy consumption is the focus and aims of this project.

# Technical Chapters (change this to something appropriate)

**Note: This part of the dissertation will normally be expanded to be *a series of chapters*.**

The technical body of the dissertation consists of a number of chapters (just one here, but there will usually be more). Follow a logical structure in how you present your work. This may be the phases of the software development cycle, or different aspects of the system's design, etc. ***However, please do not write your dissertation to read like a diary.***

The emphasis should be on requirements, design and evaluation, with implementation details being of lesser importance. The requirements should be clearly stated, following from the client needs and weaknesses identified in the state-of-the-art review. The design should include discussion of the choices that were available and why particular decisions were made.

Include a chapter demonstrating what you have achieved and how your system is used in practice – for example showing a typical session as a series of pasted in screenshots, with an accompanying commentary.

If relevant, you ***must*** also include a chapter explaining how you obtained feedback from your “customer” or potential users of your system, what feedback you actually obtained, and your analysis and comments. The evaluation should relate back to the requirements, and discuss the extent to which these were met.

Include TTD

Include spring

Toolkit then GA etc

To begin with, I created a software application that was similar to the GIN code []. However, I created the application using a new structure and different edits.

The edits I created were looking at the impact of moving large chunks of the source code and seeing if that had an impact. Other edits I created were inserting lines particularly looking at return statements and break statements. The other edits I created were using if statements and changing those.

When I created the gin code, I used Test Driven Development [REFERENCE] because …. I also used spring framework as it is a framework that I have used before and I personally found it easier to install edits.

I started with a random search then moved to local search/hill-climbing/ GA.

I used the test runner and Opacitor as a measure of fitness.

I also looked at the difference in recursive and iterative loops.

## First Section

Subdivide your text into sections.

### First Subsection

If necessary, also use subsections. Subsections are entered using the *Heading 3* paragraph style (all these heading styles are self-numbering).

#### First Subsubsection

If you really need subsubsections, enter these using the *Heading 4* paragraph style.

### Second Subsection

And, as required, more subsections.

## Second Section

As an example of a figure, consider Figure 1. Captions are entered using the *Figure* paragraph style. The figure below is placed in a *Body Centre* paragraph, which is set up in this document to have an automatic *Figure* paragraph following it. *Figure* has automatic figure numbering, and it is possible to make *cross-references* to figures. Move large figures to the top of the next page, *past any other text,* rather than having a big gap in the text.

diss-fig

1. Highly Technical Diagram

# Conclusion

## Summary

Summarise what you have achieved.

## Evaluation

Stand back and evaluate what you have achieved and how well you have met the objectives. Evaluate your achievements against your objectives in section 1.2. Demonstrate that you have tackled the project in a professional manner.

(The previous paragraph demonstrates the use of automatic cross-references: The “1.2” is a *Cross-reference* to the text in a numbered item of the document, it is *not* literal text but a *field.* The number that appears here will change automatically if the number on the referred-to section is altered, for example, if a chapter or section is added or deleted before it. Cross-references are entered using Word's **Insert** menu. Cross-references are set to update automatically when printed, but may not do so on-screen beforehand; you can update a field manually on-screen by right-clicking on it and selecting Update field from the pop-up menu.)

## Future Work

Explain any limitations in your results and how things might be improved. Discuss how your work might be developed further. Reflect on your results in isolation and in relation to what others have achieved in the same field. This self-analysis is particularly important. You should give a critical evaluation of what went well, and what might be improved.

References

Use the *Reference* paragraph style to enter and cross-reference document references. You should use the IEEE referencing style (see details on module web pages). Books [1], standards [2], reports [3], journal articles [4], conference papers [5], and web pages [6] are conventionally presented in slightly different ways.

1. Greene, D. and Williams, P. C. *Linear Accelerators for Radiation Therapy*, Second Edition. IOP Publishing Ltd., Bristol and Philadelphia, 1997.
2. ISO. *Language Of Temporal Ordering Specification*, ISO 8807, International Organization for Standardization, Geneva, 1989.
3. Jacobson, J. and Andersen, O., editors. *Software Controlled Medical Devices*. SP Report 1997:11, Swedish National Testing and Research Institute, Sweden, 1997.
4. Turner, K. J. The Rules for Sailing Races on PDAs, *J. Navigation*, 23(5):114-240, May 2002.
5. Ji, H. and Turner, K. J. Specification and Verification of Synchronous Hardware using LOTOS. In Wu, J. Chanson, S. T. Gao, Q. editors, *Proc. Formal Methods for Protocol Engineering and Distributed Systems* (FORTE XII/PSTV XIX), pages 295-312, Kluwer Academic Publishers, London, UK, October 1999.
6. The University of Stirling. *Computing Science and Mathematics Research Home Page*, http://www.cs.stir.ac.uk/research, Accessed April 2012.

Appendix 1

You may have one or more appendices containing detail, bulky or reference material that is relevant though supplementary to the main text: perhaps additional specifications, tables or diagrams that would distract the reader is placed in the main part of the dissertation. Make sure that you place appropriate cross-references in the main text to direct the reader to the relevant appendices.

*Note that you must* ***not*** *include your program listings as an appendix or appendices*. You should submit such material to the project *digital repository*.

Appendix 2 – User guide

If you produced software that is intended for others to use, or that others may wish to extend/improve, then it is advisable to include user guide and installation guide appendices.

Appendix 3 – Installation guide

If you produced software that is intended for others to use, or that others may wish to extend/improve, then it is advisable to include user guide and installation guide appendices.