CASE STUDIES IN DDD

CASE STUDIES IN DOCUMENT DRIVEN DESIGN OF SCIENTIFIC COMPUTING SOFTWARE

By

THULASI JEGATHEESAN

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements

for the degree

Master of Science in Computer Science

McMaster University

MASTER OF SCIENCE (2016) McMaster University

(Computer Science) Hamilton, Ontario

TITLE: Case Studies in DDD

AUTHOR: Thulasi Jegatheesan, B.Sc. (McMaster University)

SUPERVISORS: Dr. Spencer Smith and Dr. Ned Nedialkov

NUMBER OF PAGES: xiii, 158

Abstract

The use and development of Scientific Computing Software (SCS) has become commonplace in many fields. It is used to motivate decisions and support scientific research. Software Engineering (SE) practices have been shown to improve software quality in other domains, but these practices are not commonly used in Scientific Computing (SC). Previous studies have attributed the infrequent use of SE practices to the incompatibility of traditional SE with SC development. In this research, the SE development process, Document Driven Design (DDD), and SE tools were applied to SCS using case studies.

Five SCS projects were redeveloped using DDD and SE best practices. Interviews with the code owners were conducted to assess the impact of the redevelopment. The interviews revealed that development practices and the use of SE varied between the code owners. After redevelopment, the code owners agreed that a systematic development process can be beneficial, and they had a positive or neutral response to the software artifacts produced during redevelopment. The code owners, however, felt that the documentation produced by the redevelopment process requires too great a time commitment. To promote the use of SE in SCS development, SE practices must integrate well with current development practices of SC developers and not disrupt their regular workflow. Further research in this field should encourage practices that are easy to adopt by SC developers and should minimize the effort required to produce documentation.

Acknowledgments

I would like to express my gratitude to my supervisors, Dr. Spencer Smith and Dr. Ned Nedialkov for their continual support and insight. I am extremely grateful for the patience, advice and knowledge they provided.

I would like to thank Dr. Diane Kelly and Dr. Marilyn Lightstone for their contributions to this research, as well as the participants in this study. I would also like to thank the undergraduate student researchers for all of their hard work on the redevelopment projects.

I would like to express my heart-felt gratitude to my family, especially my parents, Ammah and Appah, for their love and support throughout my academic career, as well as Ammammah, Annah and Meera.

Above all, I thank my husband, Dan. Your unconditional love and endless encouragement made this possible.

Contents

Abstract iii

Acknowledgments iv

1 Introduction 1

1.1 Research Context 2

1.2 Motivation of Research 6

1.3 Research Problem and Scope 8

1.4 Organization of Thesis 9

2 Background 11

2.1 Desirable Qualities for Scientific Computing Software and Documentation . 11

2.2 Developing Scientific Computing Software 20

2.2.1 Challenges in Developing Scientific Computing Software 20

2.2.2 Current Practices in Scientific Computing 23

2.3 Document-Driven Design for Scientific Computing Software 25

2.3.1 Software Requirements Specification 27

2.3.2 Verification and Validation Plan 28

2.3.3 Module Guide 33

2.3.4 Module Interface Specification 34

2.3.5 Verification and Validation Report 35

2.3.6 Recommended Use of Software Engineering Tools and Practices . . 35

3 Solar Water Heating System Pilot Study 37

3.1 Software Requirements Specification for SWHS 38

3.2 Verification and Validation Plan for SWHS 49

3.3 Module Guide for SWHS 52

3.4 Verification and Validation Report for SWHS 57

3.5 Matlab Implementation 58

4 Experimental Study 61

4.1 Case Study Design and Data Collection 61

4.1.1 Criteria for a Candidate Redevelopment Project 62

4.1.2 Redevelopment and Data Collection Methods 63

4.2 Overview of Selected Case Studies 69

4.2.1 SWHS 69

4.2.2 Astro 71
4.2.3 Glass 72
4.2.4 Soil 73
4.2.5 Neuro 74
4.2.6 Acoustic 75
4.2.7 GamePhysics 76
4.3 Participant Responses 76
4.3.1 SWHS 77
4.3.2 Astro 78
4.3.3 Glass 79
4.3.4 Soil 81
4.3.5 Neuro 82

5 Discussion 84

5.1 Divergence from Intended Case Study Process 84
5.1.1 Astro 85
5.1.2 Glass 86
5.1.3 Soil 86
5.1.4 Neuro 86
5.1.5 Acoustic 87
5.1.6 GamePhysics 87

5.2 Common Participant Responses 88

5.3 Impressions from the Redevelopment Team 94

6 Conclusions **97**

6.1 Summary 98

6.1.1 Summary of Research Questions 98

6.1.2 Case Study Summary 100

6.2 Future Work 101

Bibliography **104**

A Ethics Application **116**

B Transcriptions of First Interviews **137**

C Transcriptions of Second Interviews **150**

Acronyms

API	Application Program Interface.
DDD	Document Driven Design.
FDA	Food and Drug Administration.
GUI	Graphical User Interface.
MG	Module Guide.
MIS	Module Interface Specification.
ODEs	Ordinary Differential Equations.
PCM	Phase Change Material.
SC	Scientific Computing.
SCS	Scientific Computing Software.

SE	Software Engineering.
SRS	Software Requirements Specification.
SWHS	Solar Water Heating System.
V&V	Verification and Validation.

List of Figures

1.1 The basic phases of software development [72, p.11] 4

2.1 V-model for scientific computing software documentation [65] 26

2.2 General Verification and Validation (V&V) Plan Table of Contents 32

3.1 Solar water heating tank, with heat fluxes from the coil and to the PCM of q_c and q_p, respectively 38

3.2 Table of Contents of the Software Requirements Specification for SWHS . 40

3.3 Problem Description for SWHS 42

3.4 Goal Statements for SWHS 42

3.5 Refinement of Abstract Goals to Concrete Instance Models 42

3.6 Sample Assumptions for SWHS 43

3.7 Theoretical Model 1 for SWHS 44

3.8 General Definition 2 for SWHS 45

3.9 Data Definition 1 for SWHS 46

3.10 Instance Model 1 for SWHS 47
3.11 Refinement of GS1 and GD2 to IM1 47
3.12 Data Constraints on Input Variables for SWHS 49
3.13 Likely Change 1 for SWHS 49
3.14 Traceability Matrix for Components for SWHS 50
3.15 Subset of Table of Contents for V&V Plan for Solar Water Heating System (SWHS) 51
3.16 Unit Test Information Recorded in V&V Plan for SWHS 53
3.17 Table of Contents of the MG for SWHS 54
3.18 Sample of Anticipated Changes for SWHS 54
3.19 Sample of Unlikely Changes for SWHS 55
3.20 Module Hierarchy 55
3.21 Temperature Module 55
3.22 Traceability Between Requirements and Modules 56
3.23 Uses Hierarchy among SWHS Modules 57
3.24 SWHS Matlab Control Module Excerpt 60

List of Tables

3.1 Traceability Between Modules and Matlab Files 59

4.1 Overview of Case Studies 70

4.2 Software Artifacts Delivered 70

4.3 Second Interview Summary 77

Chapter 1

Introduction

Scientific Computing (SC) consists of using computer tools to simulate mathematical models of real-world systems, with the goal of better understanding and predicting the behaviour of those systems [68]. Results produced by Scientific Computing Software (SCS) are used to motivate important decisions in a wide range of fields such as biology, climate change, urban planning, and human health and safety. There is an agreement within the Software Engineering (SE) community that following a systematic design process results in a better software product, however SE methods and technologies have not pervaded the SC community. This thesis attempts to apply SE methods and technologies to SCS, while considering the challenges that are faced by SC developers.

This chapter discusses the context, motivation, and scope of the research, and describes the organization of the thesis. Section 1.1 explains obstacles faced when applying SE methods to SC. Section 1.2 gives the motivation for this work, Section 1.3 clarifies the research problem and scope, and Section 1.4 shows the organization of the thesis.

1.1 Research Context

Alongside laboratory experiments and field studies, SCS has become an important part of the scientific process. SCS is developed for many purposes, and due to the complexity of both scientific theory and experimentation, has become necessary in many fields of research [73]. It is used to simulate physical phenomena, process large amounts of data, perform complex calculations, and address problems that are too time-intensive, expensive, or dangerous to perform experimentally [11, 28]. Scientific software can be considered another type of experimental apparatus. The results produced by SCS can be used to drive important decisions. However, SCS is often not used with the same care and rigour as laboratory and field apparatus [74]. SC developers are often unaware or unconcerned with traditional notions of software qualities and software development practices [33, 37]. In some cases, misplaced trust in SCS has resulted in high-profile retractions of scientific papers [12, 30]. More importantly, there are examples in which the misuse or failure of SCS which have more serious consequences, such as the 1991 collapse of the Sleipner A oil rig [13] and the defects found in the signal-processing software used in the oil industry to site oil wells [26].

SE is the application of a systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software [22, p.1]. SE methods have been applied successfully to many other domains since its conception in the late 1960s. The use of a well-defined model of the software development process and good analysis, design and implementation techniques can help achieve desirable software qualities [72, p.141]. A typical SE development process has well-defined phases. For instance, a generic process framework has five phases: problem definition, requirements specification, design specification, program implementation, and program maintenance [72, p.11]. A simplified view of the phases of software development is shown in Figure 1.1.

A common SE development process is the waterfall process. It became popular in the 1970's and is frequently used as a reference model in most introductory SE textbooks and industrial practices [22, p.402]. The waterfall process follows a linear cascade of phases, in which the output of each phase becomes the input of the next. The phases are separated and sequential, and each phase must be completed before proceeding to the following one. Documentation is produced in each phase and is used as input in the subsequent phase.

While the use of software engineering practices is claimed to increase desirable software qualities [28], in practice, software engineering methods are infrequently applied to the development of SCS [11, 33, 36, 58, 74]. In most application domains, software is written for economic gain and to satisfy the needs of a client. The majority of SE techniques are designed to decrease costs while increasing productivity, client satisfaction, and

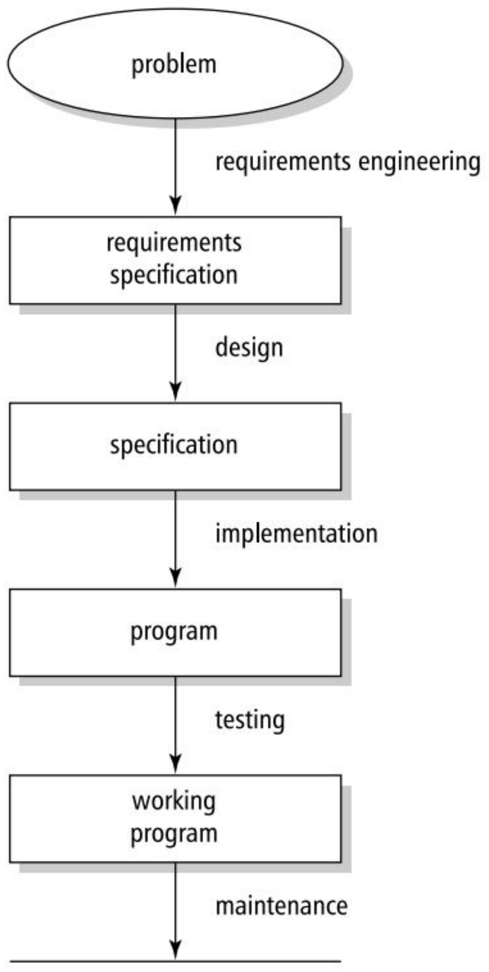


Figure 1.1: The basic phases of software development [72, p.11]

desirable software qualities. Many SC developers are professional end user developers, who are often research scientists that work in highly technical, knowledge-rich domains. They develop software to further their scientific research and goals [59]. SCS also exhibits several complications that are not usually found in other software domains. The three most notable complications are:

i the dynamic and emergent software requirements,

ii the complexity of the application domain, and

iii the lack of test oracles [62].

Many traditional SE development processes have the rigidity of the waterfall process and do not work well with the investigative nature of SC. They aim to satisfy a different set of goals and as a result, do not transfer well. At present, SCS is often written in an ad-hoc manner, following no standard development pattern [10, 60]. Previous studies that have investigated development practices in SC have discussed the incompatibility of SC development with rigid SE processes such as waterfall [35, 58, 60]. Since SC involves investigating novel scientific questions, the waterfall process often fails to support the needs of the SC developers. Agile development methods, which are more incremental and iterative, are often cited as a more appropriate process for SC developers to follow since they coincide with practices that are already present in the SC community [11, 58]. Agile methods, which emphasize working software and changing requirements, discourage comprehensive documentation [20]. The lack of documentation can be detrimental to large or complex software projects [6], and SCS, at least in terms of calculations, tends to be very complex.

Given the prevalence of the use and development of SCS, as well as the importance of the decisions that rely on it, there is an increasing need to find a repeatable and systematic way to ensure the quality of such software. In this research, we propose the use of a software development process, which will be referred to as Document Driven Design (DDD). While DDD itself does not require adherence to a rigid sequential process, it promotes the creation of documentation, as if a rational process was followed. These documents correspond to each design phase and use templates that are designed for SC. To determine the efficacy of DDD, a case study was conducted involving the redevelopment of existing SC projects using DDD. The goal of this research is to refine DDD and determine if desirable software qualities are promoted by using this development process, and SE tools and techniques.

1.2 Motivation of Research

Due to its scientific and cognitive complexity, most SCS is written by domain experts who develop the software for their own use, often referred to as professional end user developers [58]. Most SC developers are experts in their field, but typically have no prior SE education or training. Due to the complex nature of the application domain, it is difficult for external software engineers, who lack nuanced understanding of the theory and complex science of the problem, to adequately write the software. This results in most scientists writing their own software to suit their needs. While exceptions exist, in previous studies investigating the development practices in SC, it was found that collaborations between software engineers and scientists often encountered difficulties [60]. As described by Segal

[60], software engineers are used to following a traditional development process such as the waterfall process. They expect the requirements of the software to be known before the design and implementation are to be considered. However, scientists are used to the requirements emerging over time. Most scientists who develop SCS do not follow any specific development process [60, 62] as identified by the SE community. They work in an iterative and exploratory manner that does not map to a phased development process. The requirements of the software are often unknown prior to starting the implementation and emerge over time.

Documentation and a separate testing phase are also hallmarks of traditional SE processes. Since the scientists are often themselves users, and other users of the software are typically domain experts themselves, little documentation outside of high level user documentation and journal articles are produced. For end user computing, testing is often not conducted in a systematic manner [63]. SC poses difficulties with regards to testing because the correct answer of the problem is often unknown, resulting in a lack of test oracles. Additionally, SCS is usually written in an academic setting, which can results in a high turnover of developers. The absence of systematic documentation and testing, combined with the high turnover of developers, can result in difficulties reproducing computational experiments, assessing the correctness of the software, and extending and maintaining it.

Many high quality software products are produced from the SC community. However, SE methods and tools can be valuable since software processes are structured to make development systematic and less error prone [22, p.388-390], resulting in a software product that has more predictable qualities. In this research, we apply SE methods and techniques that have been tailored to the unique needs of SCS. Common SE tools and technologies are employed with the goal of producing SCS of predictable quality.

1.3 Research Problem and Scope

Generally, SCS can be divided into two categories: general purpose libraries or tools such as a solver for a system of linear equations, and software written to model physical phenomena [65]. In this study, we are primarily considering the latter, although this process and documentation is applicable to both. The projects considered for this study are of a small size due to the limited time available for redevelopment. The study participants are end user developers [59] and apart from one, the primary purpose of the considered software is to further their own needs. The software written by these scientists are used to investigate scientific, rather than computational problems. The software in question is not high performance computing software. In general, the software that is considered is not used to motivate major decisions that have an impact on health, safety or economy. However, the projects are redeveloped as if they were going to pass through a certification process.

The primary research problems that are addressed are as follows.

1. Can traditional SE methods be applied to SCS?

2. Does DDD help to achieve desirable software qualities?

3. How can DDD be refined to best fit the needs of scientists developing SCS?

To answer these research questions, DDD was applied to existing SCS. The value of DDD was assessed using interview with the code owners prior and post development. The results of the interviews were used to draw conclusions about DDD and to determine successes and avenues for further improvement.

1.4 Organization of Thesis

The thesis is organized into six chapters and three appendices.

- Chapter 1 provides an introduction to the challenges faced when applying SE methods to the development of SCS. This chapter presents the research context, motivation of this research, and its scope.

- Chapter 2 defines desirable software qualities, presents the current practices in SC, and gives an overview of DDD for SC.

- Chapter 3 describes the pilot study that was used to shape the case study design. The documentation produced is described in detail.

- Chapter 4 contains the design of the case studies, an overview of each selected case study, and the results of the interviews.

- Chapter 5 is an analysis of the results of the interviews.
- Chapter 6 provides a summary of the thesis as well as future work.
- Appendix A contains the ethics application.
- Appendix B contains the results of the first interviews.
- Appendix C contains the results of the second interviews.

Note: All documents related to the pilot study are not included in the Appendices. The complete set of pilot study documentation can be found at: https://github.com/smiths/swhs.

Chapter 2

Background

This chapter gives a brief summary of current practices in Scientific Computing (SC) (Section 2.2), the desirable software qualities that are emphasized in this study (Section 2.1), and an overview of Document Driven Design (DDD) for SC in Section 2.3. DDD is the development process applied in this study.

2.1 Desirable Qualities for Scientific Computing Software and Documentation

Quality is not a single measure, but rather it is divided into a number of quality factors, or qualities. From [22], software qualities can be classified as internal or external and product or process focused. An external quality is one that is observed by the users of a system. Internal qualities are visible to the software developers. The internal qualities largely deal with the structure of a system and help to achieve external qualities. The distinction between external and internal qualities can be difficult to discern due to the large overlap between the users and developers of Scientific Computing Software (SCS).

Software qualities can also be attributed to the process used to produce a software product, or to the software products themselves. The quality of the products produced is based on the quality of the process that leads to the product [72, p.109]. For this reason, software qualities apply to the development process, the documentation, the design, and the implementation, and should be considered at all stages of software development.

Depending on the context and purpose of the software, different qualities have different levels of importance. To control and manage qualities, they must be rigorously defined [72, p.141]. The next section discusses important software qualities for SCS, which will be emphasized throughout this thesis. The qualities are organized in four groups: correctness and reliability; maintainability and reusability; verifiability, validatability and reproducibility; and, other qualities, which includes understandability, traceability, completeness, and abstraction.

Correctness and Reliability

Correctness and reliability are essential qualities for SCS, but can be difficult to achieve (see Section 2.2.1). While it is impossible to build completely error free software, the use of Software Engineering (SE) methods can help to reduce errors, improving correctness and reliability of SCS.

Correctness

Correctness is an important quality for all software products. In general, software is correct if it behaves according to its stated specifications [22]. For SCS, there are two stages of correctness. The correctness of the mathematical model is validated against the real world problem, and the implementation is verified against the mathematical model [68]. The software should yield the correct results for all input data [70]. In SC, computational correctness is often cited as the most important quality for SCS [11, 37]. It is preferable that the software return an error, rather than yield incorrect results. Correctness is an external property that is improved when proven methods and processes are used [22].

Reliability

Reliability is a measure of the probability that the software performs its required function under stated conditions for a specified period of time [70]. While reliability and correctness are closely related, correctness is an absolute measure while reliability is a probabilistic one. In an ideal situation, where the requirements themselves are assumed to be correct, the set of all correct programs are a subset of the set of all reliable programs [22].

Maintainability and Reusability

Due to the nature of SC, SCS is often reused or built upon, as the mathematical models they are based on change. Software and documentation that is easy to modify promotes maintainability and reusability, which in turn promote other desirable software qualities, such as correctness and reliability, which are improved when trusted components are reused.

Maintainability

The maintainability of the documentation and implementation should be considered throughout the development process. Improving qualities such as readability, understandability, and reliability results in more maintainable software that is easy to modify and build upon. Good software practices such as detailed comments, descriptive variable and class names, modularization, and maximizing cohesion and minimizing coupling also promote maintainable software [22, 74]. During the development process, requirements that are anticipated to change should be noted. The design, implementation, and documentation should be developed in such a way that they can easily accommodate these anticipated changes. The structure of the documentation and the design should maintain the qualities of consistency, traceability and completeness [66]. Any changes made to documentation or design should minimally impact their structure.

Reusability

Reusability refers to the degree to which a software product or component can be used in another software system. Software reusability facilitates an increase in productivity, reduced development cost, and reduced implementation time. Reusability can also increase the reliability of software, since reliability can be increased by reusing trusted components. SCS especially lends itself to reuse, since program families, which are sets of programs that have common properties and predictable variabilities[51], are common in SC [68].

Verifiability, Validatability and Reproducibility

SCS is built to model and simulate abstractions of real world phenomena. For these models and simulations to be useful, we need to be confident that the mathematical model is suitable for the real-world problem, and the code is a suitable representation of the mathematical model, within the regime of use. Verification and validation are the assessments used to build confidence in SCS. Software artifacts with the quality of verifiability and validatability are key parts of the process to provide evidence of correctness and reliability of the computational results of the software [48]. Additionally, one of the main principles of scientific research is reproducibility. As with all scientific research, reproducibility in SC gives additional confidence in the results.

Verifiability

In general, software is verifiable if its properties can be verified with reasonable effort [22]. In the context of SCS, verification means ensuring that the mathematical models are being solved correctly, ie. "solving the equations right" [54]. Verification extends to all software artifacts: code, design, and all documentation should have the quality of verifiability. To verify the code, it must be shown that the computational implementation is a suitable representation of the mathematical model. Every design decision, line of code, and individual requirement should be clear, unambiguous, and testable to help verify that the software produced meets the requirements [66]. Traceability between the theory, numerical algorithms, and the implementation helps to achieve verifiability. The internal quality of understandability of the software artifacts also contribute to improved verifiability.

Validatability

Validation refers to the process of determining whether the mathematical model is an accurate representation of the real world for the scientific or engineering problem begin addressed by the software [48, 64], ie. "solving the right equations" [54]. Validation often involves the comparison of the computational model to experimental data. Several validation metrics are available to quantitatively compare computational and experimental results [47].

Reproducibility

Reproducibility is the ability of SC work to be duplicated, either by the same researcher or by someone else working independently [56]. Reproducibility applies to the theory in the documentation as well the computational results of the implementation [31]. Recommendations as to how to achieve reproducibility in SC are discussed in [5]. Clear documentation that includes a specification of the theory and assumptions, as well as recording the development environment, build environment, and data used for testing helps to achieve reproducible SC research. The use of tools such version control and literate programming also aid in achieving reproducibility. For example, literate tools such as Sweave produce reports that can be automatically updated if there is a change in the data or analysis [41].

Other Qualities

Other important qualities for SCS artifacts include understandability, consistency, traceability, and completeness. These qualities are internal qualities.

Understandability

Understandability refers to the extent to which a new developer can understand the documentation, design, and source code. Understandability influences maintainability and reusability of the software and documentation. If the implementation, design, and documentation are understandable, they are easier to maintain and more likely to be reused.

Understandability also helps to achieve verifiability [22]. Understandability is not always easy to achieve in SCS, since it can be difficult to make the underlying science clear in the code.

Consistency

All artifacts related to the software should have the quality of consistency. There should be no contradiction between the documentation, the design, and the implementation. There should also be no conflict between statements within the documentation. For example, throughout all documents, the meaning of a symbol should be the same. A mathematical equation that is numerically solved in the implementation should match a mathematical equation in the documentation. Consistency between and within all artifacts of the software promote other desirable qualities in software, such as reusability, maintainability, and correctness.

Traceability

Traceability involves a mapping between the requirements, the theory, the numerical algorithms, the design, and the implementation. Traceability can be achieved using cross-referencing between components in the documentation, potentially shown via a traceability matrix. The traceability matrix shows what modifications must be made if one component is modified, which helps with maintainability. Explicit traceability between the theory, numerical algorithms, and implementation increases the verifiability of the final software product [66].

Completeness

Completeness is important for both the code and documentation. The code is complete once every requirement has been fulfilled. The documentation of the requirements is complete when all goals, functionalities, attributes, design constraints, values, data, models, symbols, terms, abbreviations, acronyms, assumptions and performance requirements of the software have been defined. Completeness requires that the code can be verified using the information in the documentation, such as traceability to requirements, and design decisions and proofs.

Abstraction

Abstraction of the documentation and the implementation should be employed. The SRS document should contain all information pertaining to the properties of the software as well as what the software must do, without specifying how this is to be achieved. For example, the SRS should contain the mathematical equations to be solved by the software. The SRS should not contain the numerical algorithms used to solve the equations, as that is a design decision [66]. Abstraction should also be used in the development of the code to deal with complexity. SC has an advantage over some other types of software for abstraction because mathematicians and theoreticians have devoted considerable effort to developing underlying abstraction.

2.2 Developing Scientific Computing Software

This section discusses the challenges faced by SC developers (Section 2.2.1) and the development practices that are commonly observed in the SC community (Section 2.2.2).

2.2.1 Challenges in Developing Scientific Computing Software

Developing SCS poses unique challenges when compared to software developed for other domains. Some of the difficulties are due to inherent characteristics of SCS, while others are due to the environments in which SCS is developed. SCS is developed to solve complex and unexplored scientific and engineering problems. The development is largely done by scientists who are experts in their scientific domain rather than by software engineers, due to the large amount of scientific knowledge required. These domain scientists often have no prior software training, so common software engineering practices are rarely applied. In addition to a lack of training in, and awareness of software engineering practices, funding for proper use of software engineering techniques may not always be available. SCS development is often financially dependent on an external funding agency, which results in projects driven by scientific objectives that usually favour the development of new software instead of extending and maintaining existing software [21, 42]. Additionally, since SCS is primarily developed to answer scientific questions, apart from correctness, achieving desirable software qualities is often not an explicit priority (see Section 2.1 for description of software qualities). This can be contrasted with traditional software development which emphasizes software quality, since it is focused on producing software to fulfil the needs of a customer [28].

The remaining challenges have to do with the nature of SC itself and the incompatibility of traditional SE methods. SCS is part of an exploratory process, in which the low-level requirements are not frequently known a priori, but rather emerge through experimentation and iteration. Many traditional software engineering methods rely on having upfront requirements and have discrete phases of requirements elicitation, design, and implementation. These methods do not adapt well to SCS, since the initial requirements are usually unstable and ambiguous. Since the process of developing SCS is iterative and exploratory, it is difficult to follow discrete development stages [60].

Due to the nature of SC, while correctness and reliability are cited as the most important qualities, they are difficult to assess [11, 37]. The difficulty comes from potentially many sources of error, and the challenge with testing SCS (testing techniques are discussed in Section 2.3.2). According to [10], there are three main places where defects can occur: in the underlying science, in the translation of the model to an algorithm, or in the translation of the algorithm to code. A mathematical model is created to represent the relevant aspects of a real world problem. The mathematical model should be a suitable representation of the real world; however, the assumptions made to produce a mathematical model can introduce errors due to lack of knowledge about the problem or error by the scientists. Translation of the model into a numerical algorithm can also introduce errors.

SC often involves taking a continuous mathematical model and discretizing it so it can be solved by a computer. Approximation error will always be present to some degree due to roundoff and truncation errors. Roundoff error occurs due to the finite nature of computer floating point arithmetic. Truncation error is the result of numerical approximations, such as error arising from truncating a Taylor series approximation. Finally, mathematical models are often solved using iterative methods that construct a sequence of approximate solutions. These iterative methods are convergent if the error goes to zero as the number of iterations gets large [23]. However, in practice, the rate of convergence and the impact of round-off errors at each step may not be accounted for. While errors of these types are often unavoidable, the degree to which they are acceptable is not always clear. Lastly, error can be introduced during implementation, when the numerical algorithm is translated into code. In addition to making correctness and reliability difficult to achieve, this last step also has an impact on the understandability of the software. The underlying science behind the code may not be obvious once it is translated into a numerical algorithm.

The three stages at which error may be introduced can be uncovered to some extent by testing the software. Checking that the model is a suitable representation of the real world is called validation (see Section 2.1). This is usually accomplished by comparing computational results against experimental results or real world measurements. These results and measurements are not always available due to them being too dangerous, too expensive, or too complex for experimentation or measurement. The introduction of errors caused by discretization of the model and translation of the model into code have to do with verification, discussed in Section 2.1, which involves confirming whether the mathematical model is solved correctly. This can be difficult to assess as well, since the true solution to the mathematical model is often unknown.

The challenges faced by developers of SCS are complex and unique. Suggestions have been made attempting to address these challenges, but no consensus or universal development model has been reached. The next section discusses development practices currently employed in SC.

2.2.2 Current Practices in Scientific Computing

Many studies have investigated development practices and the use of SE in the SC community [10, 11, 36, 35, 60, 62]. Common practices were observed and are discussed below, however systematic SE practices were infrequently used. SE practices can generally be divided into activities involved in development workflow and infrastructure that supports software development [28]. Development workflow includes design, documentation, requirements elicitation, testing, and verification and validation. Infrastructure supporting software development refers to the use of software tools to aid development. The use of these practices has not been widely observed in most SC communities.

Traditional SE development methods split development into distinct phases, as shown in Figure 1.1. Each phase results in the development of either part of the software system or something associated with the system, such as documentation [22, p.6]. It was observed that the scientists and engineers developing SCS followed a rapid, incremental and iterative development process [35, 58]. Requirements gathering was integrated with coding and software evaluation, and discrete phases, such as those found in the waterfall model, were not frequently observed. In general, for SC developers, a separate software design step was not observed [28, 61]. Most of the SC developers observed wrote software for their own use. They often assumed that a design that would suit their needs would suit the needs of any user [28]. Requirements specification documents were rarely produced, and when they were produced they generally focused on high-level requirements [28].

Several studies have claimed that testing practices used by SC developers are limited [28, 34, 37]. SC developers often view the code and the implementation of model as inseparable. Testing is used to show that the theory is correct, rather than used to uncover faults in the code [28, 34]. Validation testing is often performed by the comparison of the computational results with experimental measurements but verification testing is not widely performed [34]. Testing tends to be ad-hoc and there is a lack of unit testing and automated regression testing [34, 38].

Additionally, the use of development tools such as version control, task automation, IDEs, automated build systems, and continuous integration were rarely utilized by SC developers [11, 74]. The use of these tools can lead to considerable improvements in the quality of SCS.

Applying traditional software development processes to SCS can pose many challenges; however, the use of proper SE methods can help to realize qualities resulting in better software. With an increasing reliance on software to support research and important decision making, it is vital to ensure that SCS can be trusted.

2.3 Document-Driven Design for Scientific Computing Software

The process of DDD for SC [65] is outlined in this section. Due to the exploratory and investigative nature of SC, it is often difficult to follow a systematic or rational software design process. Since the objectives of the software can change during the development process, traditional sequential development models, such as the waterfall model may not be practical. However, it is still beneficial to “fake” a rational design process [52]. A rational process, such as the waterfall process (see Section 1.1), is a process for which each step can be shown to be the best way to get to a well defined goal [52]. This helps to preserve the desired software qualities described in Section 2.1. The reuse of a rational and standardized process can result in a more successful software product, since the process can be refined and improved over time to produce more predictable results.

While in practice the development of SCS may not follow the top down approach of a rational process, in [65] an alternative is proposed. A variation of the V-model, an extension of the waterfall model (discussed in Section 1.1), is shown in Figure 2.1. This model does not require that the development process be divided into phases, but it provides a guideline for documentation. The process in Figure 2.1 can be used to produce documentation for SCS, as if a systematic development process had been followed.

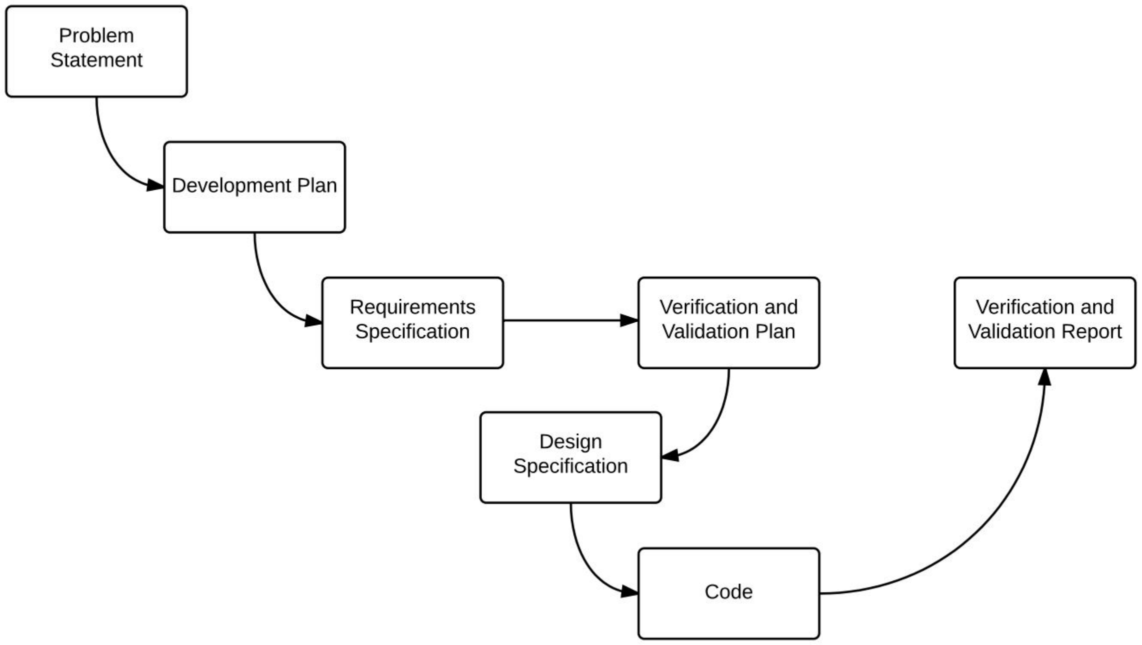

Problem Statement
Development Plan
Requirements Specification
Verification and Validation Plan
Verification and Validation Report
Design Specification
Code

Figure 2.1: V-model for scientific computing software documentation [65]

DDD includes a problem statement, requirements specification, design specification, code implementation, and a verification and validation plan and report.

2.3.1 Software Requirements Specification

The Software Requirements Specification (SRS) is the first major document in this design process. In general, an SRS provides a description of the functional and nonfunctional requirements of a software system. The purpose of the SRS is to clearly and accurately describe the characteristics and behaviour of the system. There are several advantages of creating an SRS for SCS. An SRS explicitly lays out the problem to be solved and provides a starting point for the design phase. It helps to reduce ambiguity by clearly identifying and documenting the range of model applicability, as well as the assumptions that were used to simplify the real world problem into a mathematical model. An SRS also helps to increases confidence that all cases have been considered, and encourages careful analysis of the problem prior to the design phase of the software [67].

The SRS documents created in this study use the template proposed in [67], which is intended specifically for SCS. Since SC problems are complex, there can be many sources of ambiguity when documenting the requirements. Using an SRS template is beneficial since a template identifies the information needed and acts as a checklist, reducing the chances of important information being omitted. It helps to ensure the creation of a complete SRS, which can provide a standard against which correctness and reliability of the software can be judged. The template used in this study has a hierarchical structure, which decomposes the abstract goals of the software to concrete mathematical models to be solved by the software. Since all assumptions and derivations are clearly stated, validatability and verifiability are easier to achieve. More details on the specific structure of the SRS template and the software qualities influenced can be found in Section 3.1.

2.3.2 Verification and Validation Plan

Verification and validation are integral processes in achieving software quality. The Verification and Validation (V&V) Plan document outlines the system tests and any other testing techniques that will be used to build confidence in the computational solution provided by the software.

As discussed in Section 2.2.2, developing test cases for SCS can be difficult since SC problems often lack test oracles. Traditional SE testing techniques do not always transfer well since most SE testing techniques assume accurate test oracles and the ability to run large test suites and interpret the results [39]. Since it is difficult to find meaningful test cases with known solutions, several types of testing techniques may be needed to build confidence in the software. A combination of dynamic testing and static analysis techniques can be used and recorded in the V&V Plan.

Dynamic testing is the verification of the behaviour of a program on a finite set of test cases against expected behaviour and involves execution of the code [55]. Dynamic testing techniques are listed below, with details to follow.

- system testing
- pseudo oracle
- testing known properties of the solution, ie. sanity checks
- comparison with a known solution of a subset of the real problem
- metamorphic testing
- unit testing
- integration testing
- regression testing
- mutation testing

System-level software testing is testing of the entire code as a whole [55]. System tests can be difficult to perform when a complete test oracle does not exist. In such cases, the a pseudo oracle may be used, where a pseudo oracle is an independently written program intended to fulfill the same specification as the original program [15]. While it is possible that one or both of the programs are incorrect, agreement between them helps to build confidence. It is also possible to design test cases, or sanity checks, that ensure known solution properties are satisfied. Additionally, test cases can be designed for a subset of the real problem if there is a known solution for the subset [65]. Metamorphic testing can be employed when a test oracle is not complete, which involves constructing follow-up test cases from successful test cases, with reference to certain properties of the problem known as metamorphic relations [7, 44].

Other dynamic testing methods include unit tests, integration tests, and regression tests. Unit tests cover each module of the code. They can be used to ensure that each module is working properly and allow testing to proceed iteratively throughout the development process [8]. Integration testing requires that specified test suites be run when program modules are combined into groups [55]. Regression testing involves the comparison of software output to the output from earlier versions to prevent the introduction of coding mistakes by detecting unintended consequences of changes in the code [55]. Mutation testing is a type of testing that is used to assess the quality of a set of tests. It involves generating new, slightly modified sets of the program, called mutants. The adequacy of the set of test suite are determined by the ability of the tests to detect the mutants [14].

Static analysis can also be used for verification of the code. Static analysis is any type of assessment of software correctness that does not require program execution [55]. Static analysis techniques are listed below, with details to follow.

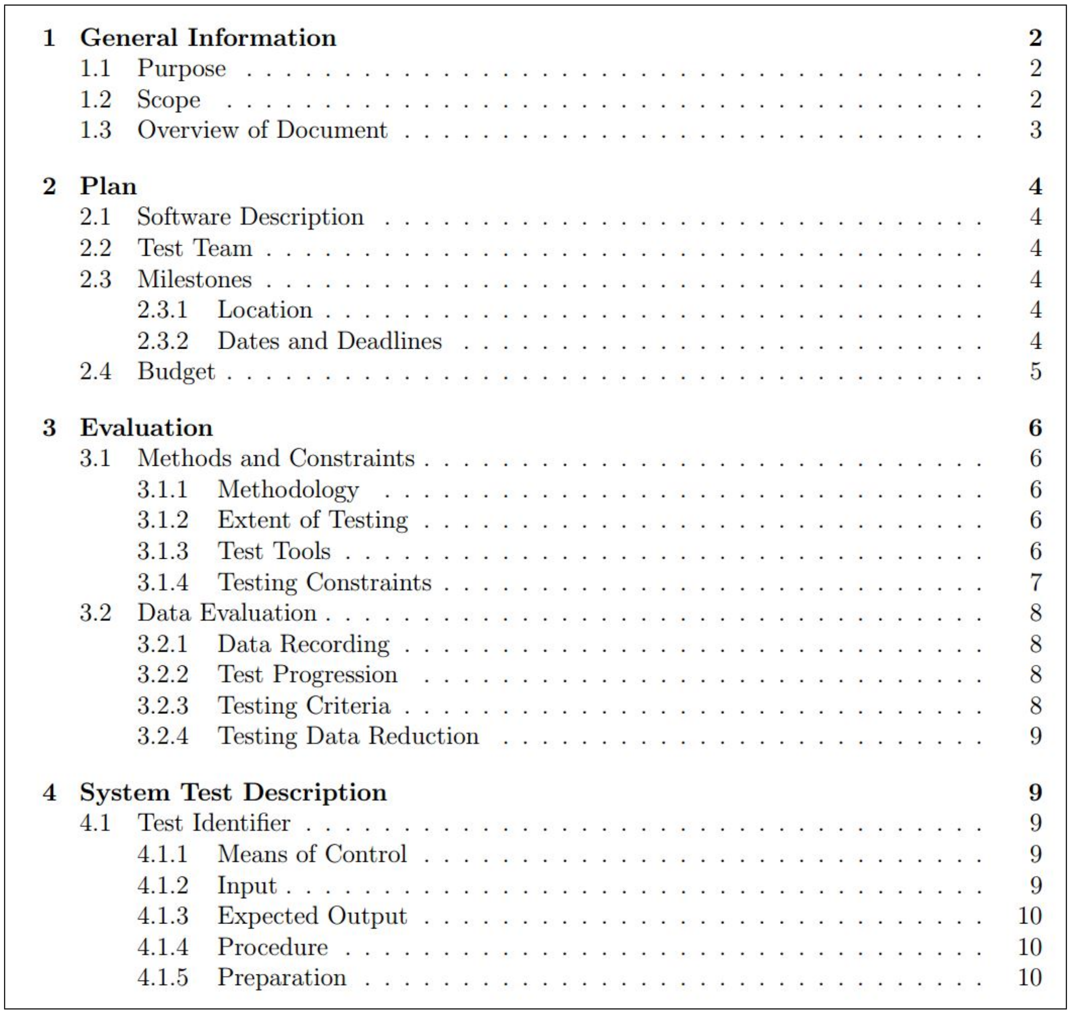
- code inspection
- code walkthroughs (peer reviews)
- automatic static analysers

- correctness proofs

Hatton [27] estimates that 40% of software failures are due to static faults. Additionally, while code review is rare in SCS development [53], many studies have shown that code review is the most effective way to find bugs [9, 17]. Code inspections involve the code developers and testers inspecting the code for errors, and code walkthroughs involve the developers explaining the code to their peers, followed by a discussion [17]. Automatic static analyzers, such as Lint [32], are external tools that are designed to find inconsistent or undefined use of a programming language that a compiler will likely overlook, as well as coding constructs that are generally considered unsafe [55].

Validation should also be included in this document. If experimental results are available, they should be identified in the V&V Plan for comparison with the computational results. Depending on the purpose of the code, a validation phase may not be necessary. For instance, for a general purpose tool like a solver for a linear system of equations. The V&V Plan should also include details about other techniques used such as code walkthroughs, code inspections, correctness proofs, and the approach for automatic testing. Additional details about the structure of the V&V Plan and examples of testing techniques can be found in Section 3.2.

Figure 2.2 shows a general case of a table of contents for the V&V Plan. The first two sections cover general and administrative information. The Evaluation section explains the methods, tools, and techniques that will be used. The System Test Description section gives an example of how a system test will be documented. In an actual V&V Plan, there would be multiple instances of this section, each corresponding to a different system test. If validation and other testing techniques are appropriate, they would be included as well.



1 General Information **2**
1.1 Purpose 2
1.2 Scope 2
1.3 Overview of Document 3

2 Plan **4**
2.1 Software Description 4
2.2 Test Team 4
2.3 Milestones 4
2.3.1 Location 4
2.3.2 Dates and Deadlines 4
2.4 Budget 5

3 Evaluation **6**
3.1 Methods and Constraints 6
3.1.1 Methodology 6
3.1.2 Extent of Testing 6
3.1.3 Test Tools 6
3.1.4 Testing Constraints 7
3.2 Data Evaluation 8
3.2.1 Data Recording 8
3.2.2 Test Progression 8
3.2.3 Testing Criteria 8
3.2.4 Testing Data Reduction 9

4 System Test Description **9**
4.1 Test Identifier 9
4.1.1 Means of Control 9
4.1.2 Input 9
4.1.3 Expected Output 10
4.1.4 Procedure 10
4.1.5 Preparation 10

Figure 2.2: General V&V Plan Table of Contents

2.3.3 Module Guide

The Module Guide (MG) is a high level design document. It provides a high level view of the software architecture and shows the decomposition of the system into modules. A module is defined as a work assignment [49] and the decomposition of the software system into modules is based on the principle of information hiding [50]. To effectively decompose a system, each module should encapsulate system changes that are likely to occur independently and should be simple enough to be understood fully. Each module is characterized by a design decision, or secret, which it hides from other modules. There should be loose coupling between modules and high cohesion within a module [45]. The interfaces between the modules should be well defined and should reveal as little as possible about its inner workings. SCS typically follows an Input $\Rightarrow$ Calculate $\Rightarrow$ Output design pattern [65]. Most designs will include an input format hiding module, an input parameter data structure hiding module, and an output format hiding module. Variation between designs will mostly be found in the modules dealing with calculations [65].

Ideally, the software architecture should be designed based on the anticipated needs of the software in the future, since early design decisions have a major impact on the quality of the final system [72, p.13]. The use of modularization in software design can improve qualities such as maintainability and understandability of software, and can reduce overall development time [45, 49]. Since a modular software system is designed with anticipated changes in mind, it should be possible to make a major software change as a set of independent changes to independent modules [50]. It is possible to modify, substitute, or remove modules with minimal impact to the rest of the software, which makes maintenance of the software easier as well as increases the likelihood of reusability of its components. Modularity also allows for increased understandability of a complex software system, since each module is designed to be comprehensible individually.

In addition to designing the software system in modules, documentation of the modules further promotes maintainability, understandability, and reusability of the code. The MG template used also promotes these qualities in the document itself. An example of an MG can be found in Section 3.3.

2.3.4 Module Interface Specification

The Module Interface Specification (MIS) is the second design document in the DDD process. It provides an abstract view of the design of the implementation; however, it is less abstract than the MG. The MIS shows the interface and external observable behaviour of each module by specifying the syntax and semantics of its access routines. It provides enough information for each module to be developed independently without specifying its internal workings, which are the secrets of the module given in the MG.

The MIS improves the understandability of the software interface and encourages reusability and maintainability of the software.

2.3.5 Verification and Validation Report

The V&V Report is the corresponding document to the V&V Plan. This document contains a summary of the results of the V&V activities and can be completed once the implementation and other documentation have been finished. The results should be summarized with enough detail to convince the reader that all V&V activities outlined in the V&V Plan were completed. This document should also contain any changes that were made to the software as a result of the issues uncovered during the V&V activities. The results of the V&V activities performed for the Solar Water Heating System (SWHS) pilot study can be found in Section 3.4.

2.3.6 Recommended Use of Software Engineering Tools and Practices

In addition to documentation, the use of SE tools can increase the efficiency and productivity of the development process, as well as improve the qualities of the software [74]. The use of software tools help to automate repetitive tasks, allow for easier debugging, and can result in code that is easier to understand, maintain, and reuse.

As described in Section 2.3.3, the code should be structured into modules. Variable and function names should be consistent, distinctive, and meaningful, and the style and formatting of the code should be consistent [74].

Version control tools, such as Apache Subversion (SVN) [19] and Git [1], should be employed for better management of all material associated with the project. Furthermore, version control ensures good data provenance, since a set of results can be tied to a specific software version and set of input files [8]. Issue tracking software should also be used to keep track of requirements and features to be added, and bugs that need to be addressed.

Testing is important throughout the development process. Test cases should be automated and can be considered prior to beginning the implementation [29]. Automated testing harnesses that include unit tests, integration tests, and regression testing make it easier to run tests frequently [74].

Chapter 3

Solar Water Heating System Pilot Study

The Solar Water Heating System (SWHS) project was used as a pilot study to help determine the structure of the case study redevelopment projects. Two informal interviews were conducted with the code owner, Dr. Marilyn Lightstone, and the results of the interview were used to shape the questions used for the subsequent case studies.

The SWHS program is a simulation of a solar water heating system incorporating Phase Change Material (PCM). The program calculates the temperature and energy of the water and PCM over time. A simple representation of the system is shown in Figure 3.1. All of the documents described in Section 2.3 were produced, except for the Module Interface Specification (MIS). A complete reimplementation and an automated testing harness was constructed as well. All documents and code produced can be found in the GitHub repository: https://github.com/smiths/swhs.

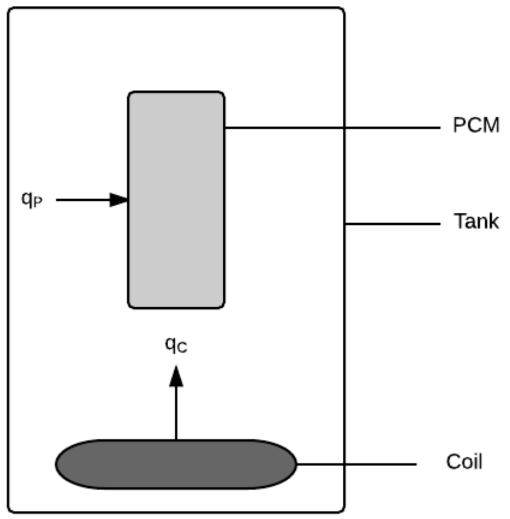
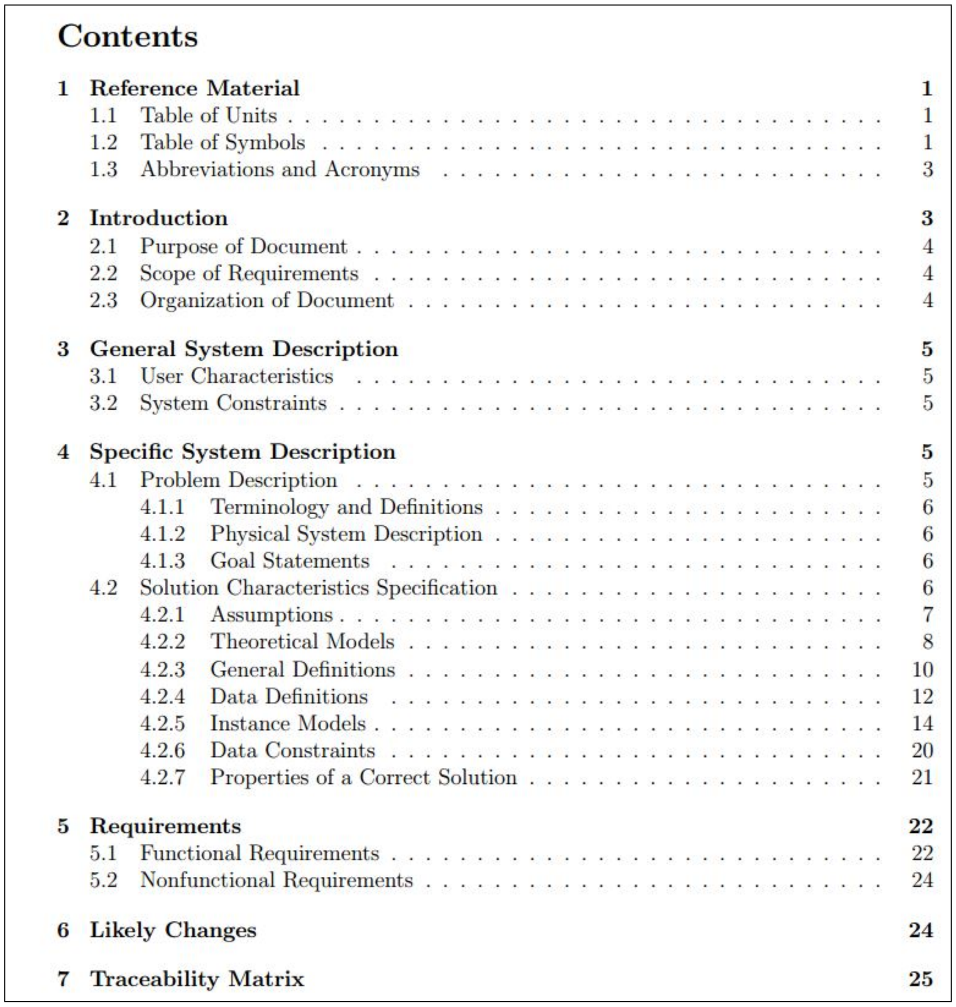


Figure 3.1: Solar water heating tank, with heat fluxes from the coil and to the PCM of q_c and q_p, respectively

3.1 Software Requirements Specification for SWHS

The different sections of the Software Requirements Specification (SRS) template are shown in Figure 3.2. The SRS has six main sections. The first section serves as a reference section, containing all units, terminology, and symbols used in the SRS. Since the meaning of a symbol or term is often dependent on the context in which it is used, a reference section with these defined can help to prevent any ambiguity. The next section gives an overview of the SRS document, including the purpose, scope, and organization of the document. The third section provides general information about the system, such as user characteristics and system constraints. This helps to ensure that the final software is used correctly. The fourth section, containing the high-level description of the problem to be solved, as well as the derivation of the mathematical model, is the most important and will be described in more detail below. The fifth section provides a concise list of the functional and nonfunctional requirements of the software. The sixth section lists the changes that are likely to be made to the mathematical model in future iterations of the software. These likely changes are taken into account when designing the implementation. The final section contains a traceability matrix that shows the dependencies between the different components of the SRS.



Contents

1 Reference Material 1
1.1 Table of Units . . . 1
1.2 Table of Symbols . . . 1
1.3 Abbreviations and Acronyms . . . 3

2 Introduction 3
2.1 Purpose of Document . . . 4
2.2 Scope of Requirements . . . 4
2.3 Organization of Document . . . 4

3 General System Description 5
3.1 User Characteristics . . . 5
3.2 System Constraints . . . 5

4 Specific System Description 5
4.1 Problem Description . . . 5
4.1.1 Terminology and Definitions . . . 6
4.1.2 Physical System Description . . . 6
4.1.3 Goal Statements . . . 6
4.2 Solution Characteristics Specification . . . 6
4.2.1 Assumptions . . . 7
4.2.2 Theoretical Models . . . 8
4.2.3 General Definitions . . . 10
4.2.4 Data Definitions . . . 12
4.2.5 Instance Models . . . 14
4.2.6 Data Constraints . . . 20
4.2.7 Properties of a Correct Solution . . . 21

5 Requirements 22
5.1 Functional Requirements . . . 22
5.2 Nonfunctional Requirements . . . 24

6 Likely Changes 24

7 Traceability Matrix 25

Figure 3.2: Table of Contents of the Software Requirements Specification for SWHS

The structure of the SRS template supports both reuse and maintenance. To help achieve both these qualities, the principle of "separation of concerns" is used. The information in the SRS is presented in decreasing levels of abstraction, with each level being self-contained. The abstract goals (in Section 4.1.3) are systematically refined into concrete mathematical models (in Section 4.2.5), with each step broken down into structured units. This structure allows for reuse and easy modification of both high-level and low-level requirements, without disrupting the structure of the SRS. Traceability, using cross-referencing throughout the document, helps maintain consistency, if any changes are made. Traceability also aids in verification, since the derivation of the mathematical model can be traced from the abstract goals to the concrete mathematical equations.

As stated above, the Specific System Description section (Section 4.0) contains a description of the problem to be solved. This section is divided into two subsections. The first subsection gives a high-level view of the problem to be solved, a description of the physical system being modelled, and the goals of the software. The problem description is a concise statement of the problem. It should be written in simple enough terms that an educated layperson would be able to understand the function of the software. The purpose of the problem description is to provide a concise statement of the problem and to narrow the scope of the problem, without specifying how to solve it. The problem description for SWHS can be found in Figure 3.3. From the problem description, abstract goal statements can be produced. The goal statements for SWHS are shown in Figure 3.4. The physical system is also described in this subsection. A diagram of the physical system modelled by SWHS is shown in Figure 3.1.

The second subsection includes the derivation of the mathematical model to be solved by the software. The mathematical model, is refined from the abstract goals as shown in

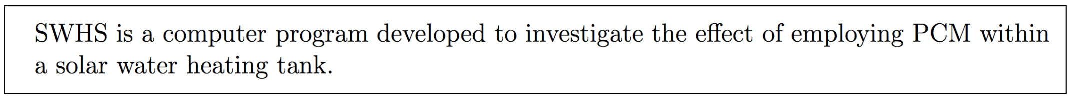
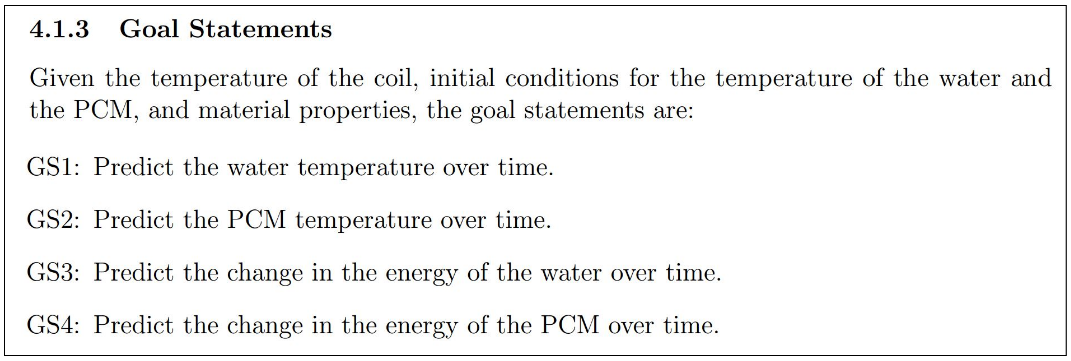

SWHS is a computer program developed to investigate the effect of employing PCM within a solar water heating tank.

Figure 3.3: Problem Description for SWHS


4.1.3 Goal Statements

Given the temperature of the coil, initial conditions for the temperature of the water and the PCM, and material properties, the goal statements are:

GS1: Predict the water temperature over time.

GS2: Predict the PCM temperature over time.

GS3: Predict the change in the energy of the water over time.

GS4: Predict the change in the energy of the PCM over time.

Figure 3.4: Goal Statements for SWHS

Figure 3.5. All assumptions, data definitions, and derivations used to construct the final model can be found here. The mathematical equations that are solved by the software are referred to as instance models.

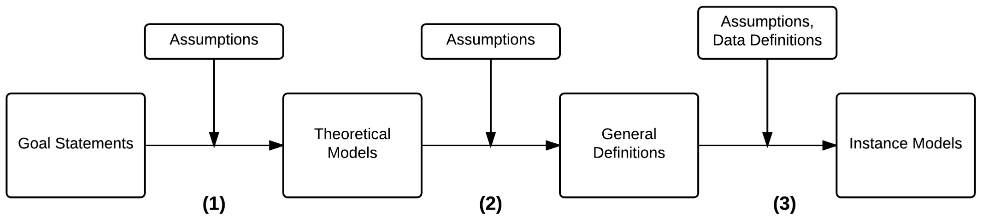


Figure 3.5: Refinement of Abstract Goals to Concrete Instance Models

Clear and correct assumptions are a crucial part of the mathematical modelling process.

Assumptions simplify the original problem into something that can be solved algorithmically. It is essential that the assumptions are defined clearly. Different assumptions can result in different mathematical models, changing the applicability of the solution to different scenarios. In the context of the SRS, the assumptions are used in the refinement of goal statements (see Figure 3.5). A sample of the assumptions from SWHS are shown in Figure 3.6.

A1: The only form of energy that is relevant for this problem is thermal energy. All other forms of energy, such as mechanical energy, are assumed to be negligible [T1].

A2: All heat transfer coefficients are constant over time [GD1].

A3: The water in the tank is fully mixed, so the temperature is the same throughout the entire tank [GD2, DD2].

A4: The PCM has the same temperature throughout [GD2, DD2, LC1].

A5: Density of the water and PCM have no spatial variation; that is, they are each constant over their entire volume [GD2].

A6: Specific heat capacity of the water and PCM have no spatial variation; that is, they are each constant over their entire volume [GD2].

Figure 3.6: Sample Assumptions for SWHS

The theoretical models are the governing equations and scientific laws that are the basis of the mathematical problem solved by the software. The goal statements are refined into theoretical models using the assumptions, as shown in step (1) of Figure 3.5. For example, GS1 and GS2 (Figure 3.4) are refined into T1, shown in Figure 3.7, using A1 (Figure 3.6). T1 contains the equations for the conservation of thermal energy, which forms the foundation for the derivation of the final instance models. The abstract nature and format of the theoretical model facilitates reuse for other problems. It is possible to reuse T1 for other thermal analysis problems. For example, T1 is also used in the SRS of another study performing the thermal analysis of a single fuelpin in a nuclear reactor [40]. Using different assumptions, T1 is refined into a different general definitions from the SWHS problem.

Number	T1
Label	**Conservation of thermal energy**
Equation	$-\nabla \cdot \mathbf{q} + g = \rho C \frac{\partial T}{\partial t}$
Description	The above equation gives the conservation of energy for time varying heat transfer in a material of specific heat capacity C and density ρ, where $\mathbf{q}$ is the thermal flux vector, g is the volumetric heat generation, T is the temperature, t is time, and ∇ is the gradient operator. For this equation to apply, other forms of energy, such as mechanical energy, as assumed to be negligible in the system (A1).
Source	http://www.efunda.com/formulae/heat_transfer/conduction/overview_cond.cfm
Ref. By	GD2

Figure 3.7: Theoretical Model 1 for SWHS

The general definitions are the laws and equations that are used indirectly to develop the mathematical models. The general definitions are refined from the theoretical models using the assumptions, as shown in step (2) of Figure 3.5. T1 (Figure 3.7) is refined into GD2 shown in Figure 3.8 using A3 to A6 (Figure 3.6).

Number	GD2
Label	**Simplified rate of change of temperature**
Equation	$mC\frac{dT}{dt} = q_{\text{in}}A_{\text{in}} - q_{\text{out}}A_{\text{out}} + gV$
Description	The basic equation governing the rate of change of temperature, for a given volume V, with time. m is the mass (kg). C is the specific heat capacity ($\mathrm{J\,kg^{-1}\,°C^{-1}}$). T is the temperature (°C) and t is the time (s). q_{in} and q_{out} are the in and out heat transfer rates, respectively ($\mathrm{W\,m^{-2}}$). A_{in} and A_{out} are the surface areas over which the heat is being transferred in and out, respectively ($\mathrm{m^2}$). g is the volumetric heat generated ($\mathrm{W\,m^{-3}}$). V is the volume ($\mathrm{m^3}$).
Ref. By	IM1, IM2

Figure 3.8: General Definition 2 for SWHS

The data definitions are the symbols and physical data used in developing the final instance models. An example, DD1 is shown in Figure 3.9.

Number	DD1
Label	**Heat flux out of coil**
Symbol	q_C
SI Units	$\mathrm{W\,m^{-2}}$
Equation	$q_C(t) = h_C(T_C - T_W(t))$, over area A_C
Description	T_C is the temperature of the coil. T_W is the temperature of the water. The heat flux out of the coil, q_C, is found by assuming that Newton's Law of Cooling applies (A7). This law (GD1) is used on the surface of the coil, which has area A_C and heat transfer coefficient h_C. This equation assumes that the temperature of the coil is constant over time (A8) and that is does not vary along the length of the coil (A9).
Sources	[4]
Ref. By	IM1

Figure 3.9: Data Definition 1 for SWHS

The instance models are the mathematical equations that the software is intended to solve. Here, the problem defined in the Problem Description section is reduced to one expressed in mathematical terms. The instance models use the concrete symbols defined in the data definitions to replace the abstract symbols in the theoretical models, step (3) in Figure 3.5. For this problem, the instance models are a system of Ordinary Differential Equations (ODEs) representing the energy balance on the water and PCM, as well as the energy equations for the water and PCM. IM1, shown in Figure 3.10, shows the energy balance on the water. IM1 is refined from GD2 using A15, A16, DD1, and DD2. A15, A16, and DD2 are not shown in this section, but can be found in the GitHub repository. Figure 3.11 shows the specific traceability to assumptions and data definitions for refining

GS1 and GS2 into IM1.

Number	IM1
Label	**Energy balance on water to find T_W**
Input	m_W, C_W, h_C, A_C, h_P, A_P, t_{final}, T_C, T_{init}, $T_P(t)$ from IM2 The input is constrained so that $T_{\text{init}} \leq T_C$ (A11)
Output	$T_W(t)$, $0 \leq t \leq t_{\text{final}}$, such that $\frac{dT_W}{dt} = \frac{1}{\tau_W}[(T_C - T_W(t)) + \eta(T_P(t) - T_W(t))]$, $T_W(0) = T_P(0) = T_{\text{init}}$ (A12) and $T_P(t)$ from IM2
Description	T_W is the water temperature (°C). T_P is the PCM temperature (°C). T_C is the coil temperature (°C). $\tau_W = \frac{m_W C_W}{h_C A_C}$ is a constant (s). $\eta = \frac{h_P A_P}{h_C A_C}$ is a constant (dimensionless). The above equation applies as long as the water is in liquid form, $0 < T_W < 100^o$C, where 0^oC and 100^oC are the melting and boiling points of water, respectively (A14).
Sources	[4]
Ref. By	IM2

Figure 3.10: Instance Model 1 for SWHS

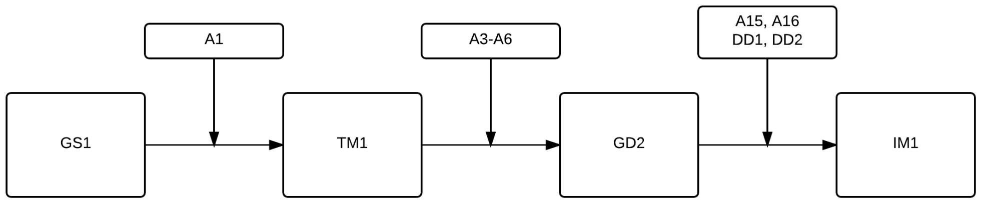
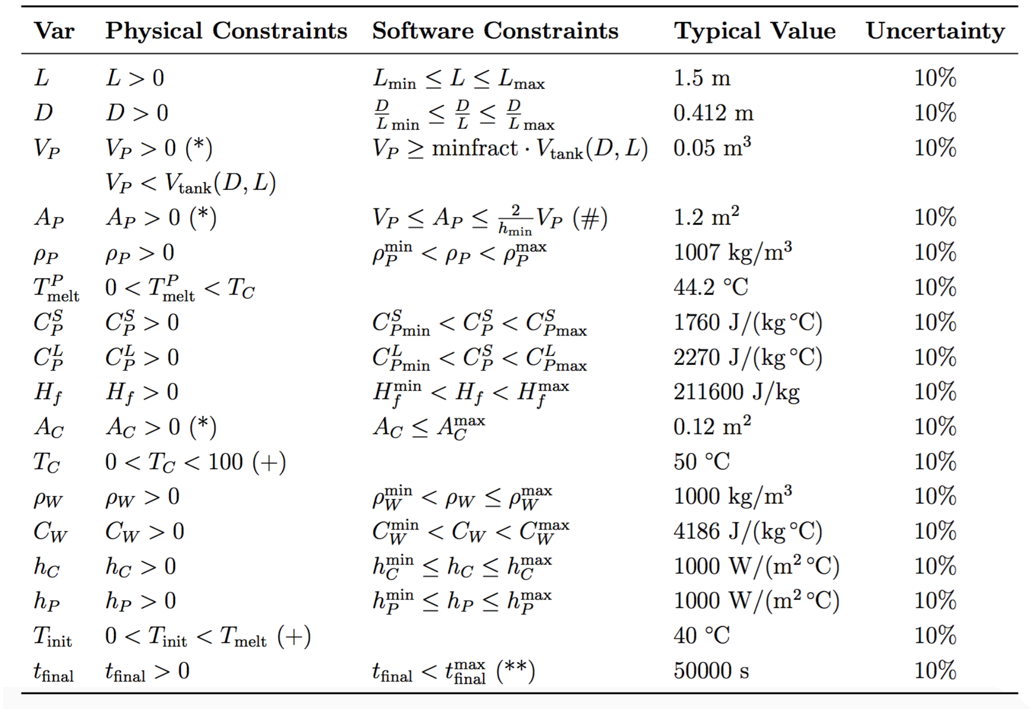


Figure 3.11: Refinement of GS1 and GD2 to IM1

Section 5 of the SRS contains the Data Constraints, which are necessary for solution validation. This section clarifies environment and system limitations imposed on admissible data, such as input parameter constraints, as shown in Figure 3.12. For example, the PCM volume must be sufficiently large to prevent a divide by zero error. The PCM volume also must be less than the volume of the tank since the PCM is contained within the tank. An explanation of the superscripts in the table are given in the SRS. Lastly, the Solution Characteristics Specification includes a section that describes any properties that a correct solution to the physical problem will have. For the physical problem modelled by SWHS, a correct solution must exhibit the law of conservation of energy. This property can be used to create additional system tests to build confidence in the solution of SWHS.

To accommodate the incremental and iterative nature of most Scientific Computing Software (SCS) development, the final section of the SRS lists the changes that are likely to be made to the model in the future. The assumptions that must be altered are shown beside each likely change (see Figure 3.13). The traceability within the document allows for easier tracking of changes since the cross-referencing shows which equations in the SRS are affected by a particular assumption.

The final section of the SRS contains the traceability matrix. The purpose of the traceability matrix is to provide an easy reference on what has to be modified if a certain component is changed. If a component shown in one of the rows is modified, every item in the corresponding column of the component marked with an “X” might also have to be



Var	Physical Constraints	Software Constraints	Typical Value	Uncertainty
L	$L > 0$	$L_{\text{min}} \le L \le L_{\text{max}}$	1.5 m	10%
D	$D > 0$	$\frac{D}{L}_{\text{min}} \le \frac{D}{L} \le \frac{D}{L}_{\text{max}}$	0.412 m	10%
V_P	$V_P > 0$ (*)	$V_P \ge \text{minfract} \cdot V_{\text{tank}}(D, L)$	0.05 m^3	10%
	$V_P < V_{\text{tank}}(D, L)$			
A_P	$A_P > 0$ (*)	$V_P \le A_P \le \frac{2}{h_{\text{min}}} V_P$ (#)	1.2 m^2	10%
ρ_P	$\rho_P > 0$	$\rho_P^{\text{min}} < \rho_P < \rho_P^{\text{max}}$	1007 kg/m^3	10%
T_{melt}^P	$0 < T_{\text{melt}}^P < T_C$		44.2 °C	10%
C_P^S	$C_P^S > 0$	$C_{P\text{min}}^S < C_P^S < C_{P\text{max}}^S$	1760 J/(kg °C)	10%
C_P^L	$C_P^L > 0$	$C_{P\text{min}}^L < C_P^S < C_{P\text{max}}^L$	2270 J/(kg °C)	10%
H_f	$H_f > 0$	$H_f^{\text{min}} < H_f < H_f^{\text{max}}$	211600 J/kg	10%
A_C	$A_C > 0$ (*)	$A_C \le A_C^{\text{max}}$	0.12 m^2	10%
T_C	$0 < T_C < 100$ (+)		50 °C	10%
ρ_W	$\rho_W > 0$	$\rho_W^{\text{min}} < \rho_W \le \rho_W^{\text{max}}$	1000 kg/m^3	10%
C_W	$C_W > 0$	$C_W^{\text{min}} < C_W < C_W^{\text{max}}$	4186 J/(kg °C)	10%
h_C	$h_C > 0$	$h_C^{\text{min}} \le h_C \le h_C^{\text{max}}$	1000 W/(m^2 °C)	10%
h_P	$h_P > 0$	$h_P^{\text{min}} \le h_P \le h_P^{\text{max}}$	1000 W/(m^2 °C)	10%
T_{init}	$0 < T_{\text{init}} < T_{\text{melt}}$ (+)		40 °C	10%
t_{final}	$t_{\text{final}} > 0$	$t_{\text{final}} < t_{\text{final}}^{\text{max}}$ (**)	50000 s	10%

Figure 3.12: Data Constraints on Input Variables for SWHS

LC1: A4 - PCM is actually a poor thermal conductor, so the assumption of uniform PCM temperature is not likely.

Figure 3.13: Likely Change 1 for SWHS

modified. A subset of the full traceability matrix for SWHS is shown in Table 3.14.

3.2 Verification and Validation Plan for SWHS

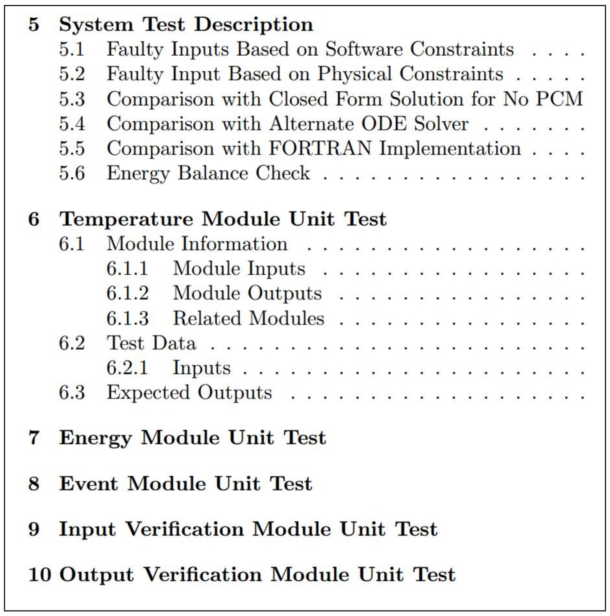
The Verification and Validation (V&V) Plan contains all of the verification and validation activities to be carried out for SWHS. Six system tests were carried out as well as unit

	T1	T2	T3	GD1	GD2	DD1	DD2	DD3	DD4	IM1	IM2	IM3	IM4
T1													
T2			X										
T3													
GD1													
GD2	X												
DD1				X									
DD2				X									
DD3													
DD4								X					
IM1					X	X	X				X		
IM2					X		X		X	X			X
IM3		X											
IM4		X	X				X	X	X		X		

Figure 3.14: Traceability Matrix for Components for SWHS

testing of the modules. Figure 3.15 shows an excerpt from the table of contents of the V&V Plan for SWHS. The first two sections outline the tests for checking the system behaviour when faulty inputs are encountered. These tests were derived using the Data Constraints table from the SRS shown in Figure 3.12. The system is expected to produce either an error message or a warning message for faulty input. A specific error message exists for each row of the Physical Constraint in Figure 3.12 and a specific warning message exists for each Software Constraint in Figure 3.12. The expected message is recorded in the V&V Plan.

A common difficulty with Scientific Computing (SC) is the lack of test oracles (See Section 2.2.1). Since no true solution is available for the mathematical model solved by SWHS, different testing techniques were used to build confidence in the numerical solution.



5 System Test Description
- 5.1 Faulty Inputs Based on Software Constraints . . . .
- 5.2 Faulty Input Based on Physical Constraints . . . . .
- 5.3 Comparison with Closed Form Solution for No PCM
- 5.4 Comparison with Alternate ODE Solver . . . .
- 5.5 Comparison with FORTRAN Implementation . . . .
- 5.6 Energy Balance Check . . . .

6 Temperature Module Unit Test
- 6.1 Module Information . . . .
  - 6.1.1 Module Inputs . . . .
  - 6.1.2 Module Outputs . . . .
  - 6.1.3 Related Modules . . . .
- 6.2 Test Data . . . .
  - 6.2.1 Inputs . . . .
- 6.3 Expected Outputs . . . .

7 Energy Module Unit Test

8 Event Module Unit Test

9 Input Verification Module Unit Test

10 Output Verification Module Unit Test

Figure 3.15: Subset of Table of Contents for V&V Plan for SWHS

The next four system tests compare the solution produced by SWHS with solution produced by considering a subset of the mathematical model, with other computational solutions, and by checking known properties of the solution.

Although a true solution is not available, it is possible to find a solution to a special case of the mathematical model. A closed form solution was found for the mathematical model if the PCM volume is zero. While SWHS cannot accept a PCM volume of zero, since it would result in a divide by zero error, the volume can be set to a negligible amount (i.e. 10^{-6}). The numerical solution produced is compared to the closed form solution in a test case.

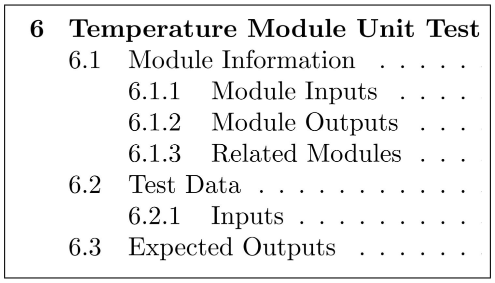
The next two system tests involve comparing the results of SWHS to the other numerical solutions. The first test replaces the ODE solver used by SWHS with a solver that uses a different numerical method to compute the solution. The second test compares the results produced by SWHS with an independent FORTRAN implementation that solves the same mathematical model.

The final test case does not test the results of the numerical solution of SWHS but verifies a property the true solution is known to have. For the solution to be correct, energy must be conserved through the system. This test is designed to verify that the total energy through the coil is equal to the total energy in the system, the total energy output subtracted from the total energy added through the coil equals the total energy in the water, and the total energy input to the PCM is equal to the energy in the PCM. These tests verify that the solution exhibits the law of conservation of energy, which must be true for the solution to be correct.

The final set of tests are unit tests. The information recorded for each unit test is shown in Figure 3.16. The expected output of the module was calculated by hand and recorded in the V&V Plan for comparison.

3.3 Module Guide for SWHS

The modular decomposition for SWHS is recorded in the Module Guide (MG). Figure 3.17 shows the sections of the MG. This document begins with an explicit statement of antici-



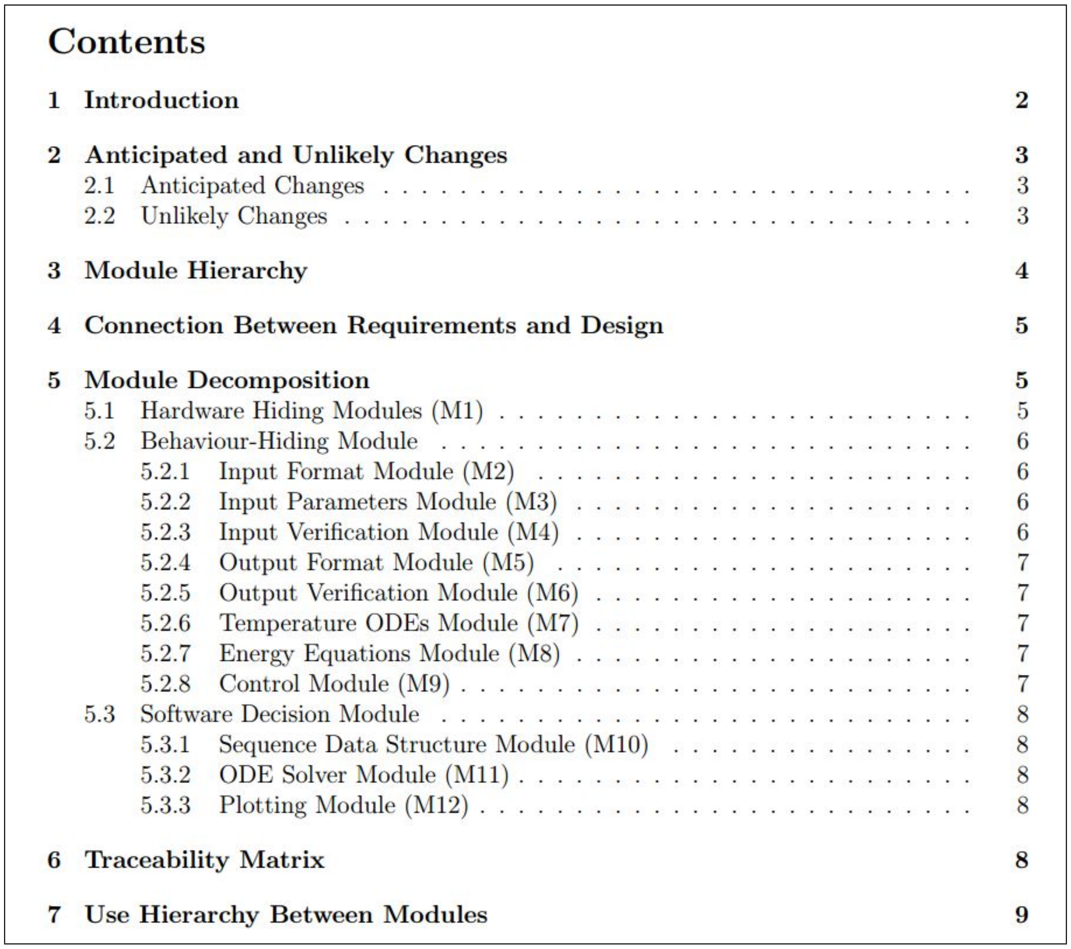
6 Temperature Module Unit Test
- 6.1 Module Information . . . . .
  - 6.1.1 Module Inputs . . . .
  - 6.1.2 Module Outputs . . .
  - 6.1.3 Related Modules . . .
- 6.2 Test Data . . . .
  - 6.2.1 Inputs . . . .
- 6.3 Expected Outputs . . . . . .

Figure 3.16: Unit Test Information Recorded in V&V Plan for SWHS

pated changes and unlikely changes, an example of which are shown in Figure 3.18. Each anticipated change corresponds to one module. If an anticipated change must be made, only one module needs to be reimplemented. An excerpt of the unlikely changes are shown in Figure 3.19. These changes were not considered when designing the system and would likely require changing more than one module.

The modules are summarized in a decomposition of secrets hierarchy, as shown in Figure 3.20. The modules in Level 2 are the leaves in the hierarchy tree and so are the modules that will actually be implemented.

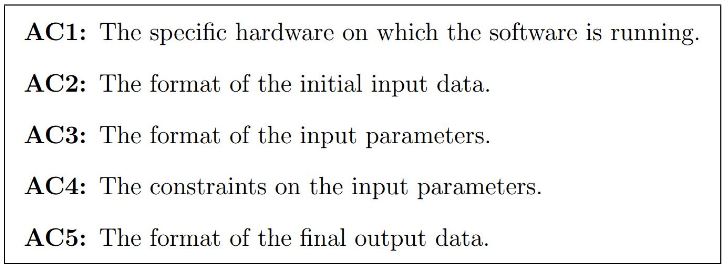
Each module is decomposed according to the principle of information hiding discussed in Section 2.3.3, proposed by [50]. The details of the Temperature Module are shown in Figure 3.21. The decomposition describes the *Secrets* and *Services* of the module, where the *Secrets* are the design decisions hidden by the module and the *Services* are a statement of the function provided by the module. The decomposition suggests how to implement the module, which depends on the programming language that is selected for the implementa-



Contents

1 Introduction	**2**
2 Anticipated and Unlikely Changes	**3**
2.1 Anticipated Changes	3
2.2 Unlikely Changes	3
3 Module Hierarchy	**4**
4 Connection Between Requirements and Design	**5**
5 Module Decomposition	**5**
5.1 Hardware Hiding Modules (M1)	5
5.2 Behaviour-Hiding Module	6
5.2.1 Input Format Module (M2)	6
5.2.2 Input Parameters Module (M3)	6
5.2.3 Input Verification Module (M4)	6
5.2.4 Output Format Module (M5)	7
5.2.5 Output Verification Module (M6)	7
5.2.6 Temperature ODEs Module (M7)	7
5.2.7 Energy Equations Module (M8)	7
5.2.8 Control Module (M9)	7
5.3 Software Decision Module	8
5.3.1 Sequence Data Structure Module (M10)	8
5.3.2 ODE Solver Module (M11)	8
5.3.3 Plotting Module (M12)	8
6 Traceability Matrix	**8**
7 Use Hierarchy Between Modules	**9**

Figure 3.17: Table of Contents of the MG for SWHS



AC1: The specific hardware on which the software is running.

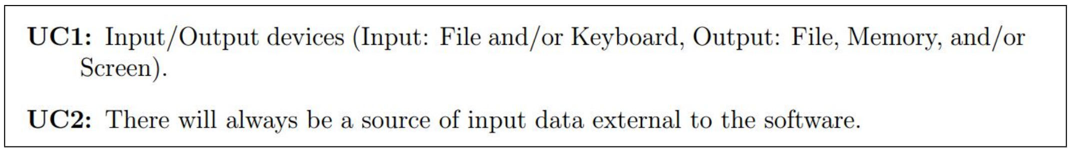
AC2: The format of the initial input data.

AC3: The format of the input parameters.

AC4: The constraints on the input parameters.

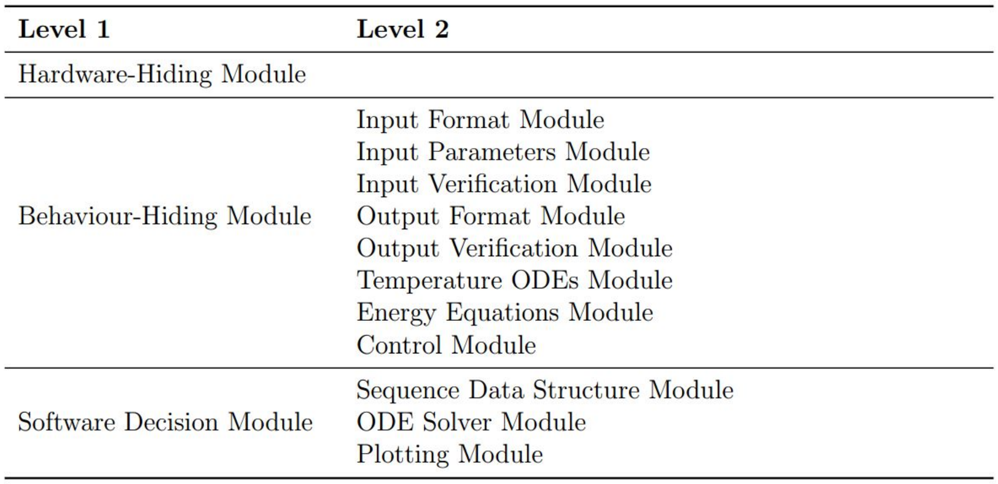
AC5: The format of the final output data.

Figure 3.18: Sample of Anticipated Changes for SWHS


UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: There will always be a source of input data external to the software.

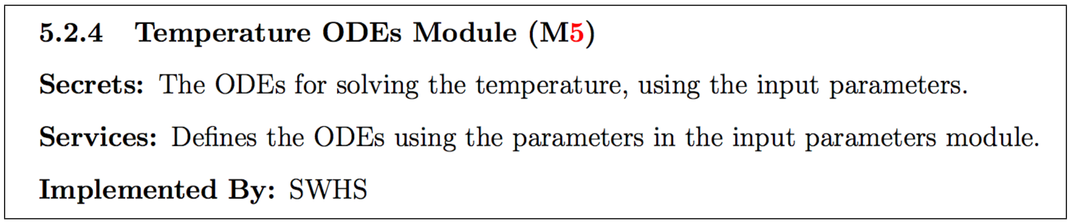
Figure 3.19: Sample of Unlikely Changes for SWHS



Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Format Module Input Parameters Module Input Verification Module Output Format Module Output Verification Module Temperature ODEs Module Energy Equations Module Control Module
Software Decision Module	Sequence Data Structure Module ODE Solver Module Plotting Module

Figure 3.20: Module Hierarchy

tion.


5.2.4 Temperature ODEs Module (M5)

Secrets: The ODEs for solving the temperature, using the input parameters.

Services: Defines the ODEs using the parameters in the input parameters module.

Implemented By: SWHS

Figure 3.21: Temperature Module

The MG also contains two traceability matrices. The first, which is shown in Figure 3.22, gives the mapping between the modules and the requirements. Should any requirements change, changes will likely need to be made to each module listed. For example, R3 states that the software must verify that the inputs satisfy the required physical constraints that are shown in Figure 3.12. Should this requirement change, only M4, the Input Verification Module would need to be modified. The second traceability matrix shown in the MG gives the mapping between the anticipated changes and the modules. As mentioned previously, for SWHS, a one-to-one mapping exists.

Req.	**Modules**
R1	M1, M2, M3, M9
R2	M2, M3
R3	M4
R4	M5, M9
R5	M5, M7, M9, M10, M11, M12
R6	M5, M7, M9, M10, M11, M12
R7	M5, M8, M9, M10, M12
R8	M5, M8, M9, M10, M12
R9	M6
R10	M5, M7, M9
R11	M5, M7, M8, M9

Figure 3.22: Traceability Between Requirements and Modules

Lastly, the MG contains a directed acyclic graph showing the relation between the modules. This is shown in the use hierarchy in Figure 3.23. As mentioned in Section 2.3.3, each level of the hierarchy offers a testable and usable subset of the system. The modules in the higher levels are essentially simpler because they use the modules in the lower levels to implement their functionality.

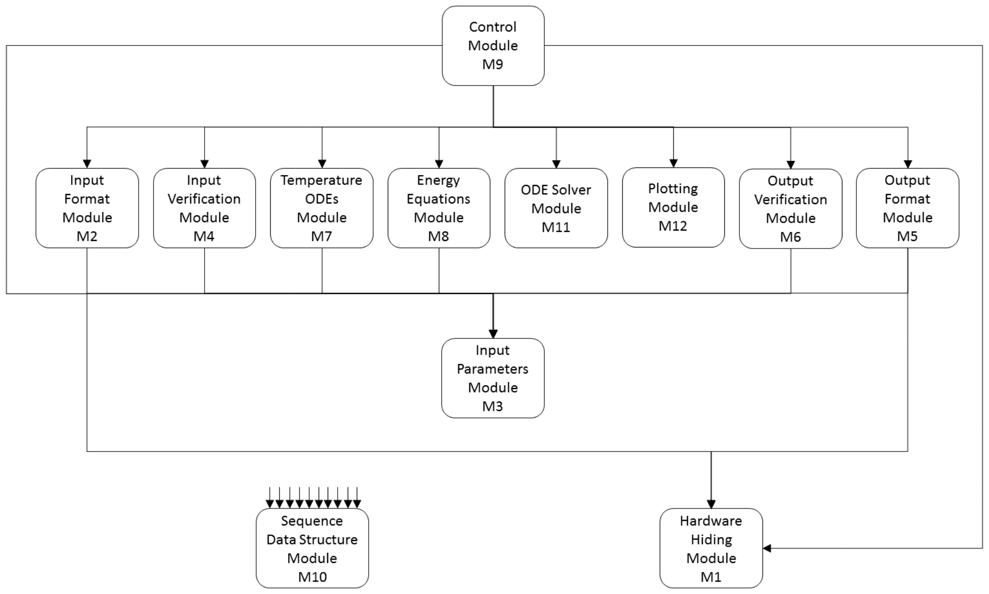

Control Module M9
Input Format Module M2
Input Verification Module M4
Temperature ODEs Module M7
Energy Equations Module M8
ODE Solver Module M11
Plotting Module M12
Output Verification Module M6
Output Format Module M5
Input Parameters Module M3
Sequence Data Structure Module M10
Hardware Hiding Module M1

Figure 3.23: Uses Hierarchy among SWHS Modules

3.4 Verification and Validation Report for SWHS

The final document produced for the SWHS project was the V&V Report. This document summarizes the results of the testing activities that were outlined in the V&V Plan (see Section 3.2). The V&V Report records whether each test passed based on the expected outputs and tolerances defined in the V&V Plan.

All of the tests conducted for SWHS passed based on the criteria given in the V&V Report. No changes were made to the software as a result of the testing.

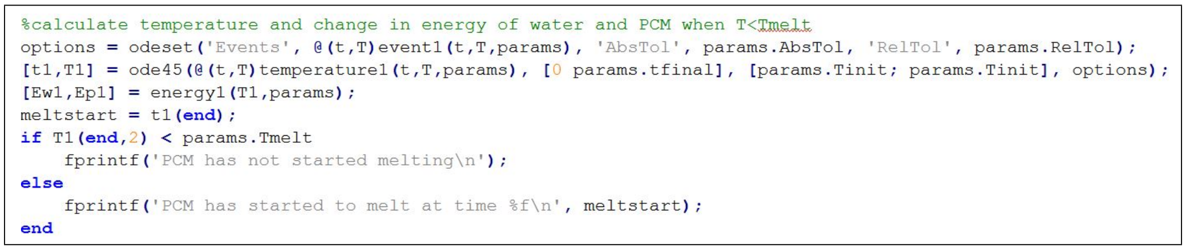
3.5 Matlab Implementation

The Matlab implementation differed from the original FORTRAN implementation in several ways. The original implementation was not divided into modules. The program accepted a text file as input and produced a text file as output. The Matlab implementation is split into modules as shown in Section 3.3. Table 3.1 shows the traceability between the modules and the Matlab files. The Matlab program was designed with change in mind, as described in Section 3.3. An attempt to minimize the amount of alteration of the Matlab files if any anticipated changes were required. Two examples are given below.

In the FORTRAN implementation, the Euler method is used to solve the ODEs and is hard-coded into the program. Different solvers may be required if the mathematical model is altered. Changing the numerical algorithm for solving the ODEs would require extensive modifications of the code. In the Matlab implementation, the built-in solver, ode45, is used, which uses the explicit Runge-Kutta methods. Changing the solver requires minimal modification of the Control Module, M9, as shown in Figure 3.24. Additionally, the Matlab implementation creates an output file that contains both input and output parameters. To add a parameter, only two files must be modified: `load_params` and `verify_params`.

Modules	**Matlab Filename(s)**	
M1	Hardware Hiding Module	Implemented by OS
M2	Input Format Module	`load_params`
M3	Input Parameters Module	`load_params`
M4	Input Verification Module	`verify_params`
M5	Output Format Module	`output`, `plot`
M6	Output Verification Module	`verify_output`
M7	Temperature ODEs Module	`temperature1`, `temperature2`, `temperature3`, `event1`, `event2`
M8	Energy Equations Module	`energy1`, `energy2`, `energy3`
M9	Control Module	`main`
M10	Sequence Data Structure Module	Implemented by Matlab
M11	ODE Solver Module	Implemented by Matlab
M12	Plotting Module	Implemented by Matlab

Table 3.1: Traceability Between Modules and Matlab Files



```
%calculate temperature and change in energy of water and PCM when T<Tmelt
options = odeset('Events', @(t,T)event1(t,T,params), 'AbsTol', params.AbsTol, 'RelTol', params.RelTol);
[t1,T1] = ode45(@(t,T)temperature1(t,T,params), [0 params.tfinal], [params.Tinit; params.Tinit], options);
[Ew1,Ep1] = energy1(T1,params);
meltstart = t1(end);
if T1(end,2) < params.Tmelt
    fprintf('PCM has not started melting\n');
else
    fprintf('PCM has started to melt at time %f\n', meltstart);
end
```

Figure 3.24: SWHS Matlab Control Module Excerpt

Chapter 4

Experimental Study

A qualitative case study approach is used to study the impact of traditional Software Engineering (SE) practices on Scientific Computing Software (SCS). The design of the case study, the method of data collection, and a summary of the participant responses can be found in this chapter. Details on the ethics application can be found in Appendix A and transcripts of the interviews can be found in Appendices B and C. Identifying information has been redacted from the interview transcripts.

4.1 Case Study Design and Data Collection

The case study design and data collection methods are described here. Section 4.1.1 contains the criteria used to determine whether a project was appropriate for this study. Scientific Computing (SC) projects that fit the described criteria were selected and redeveloped by undergraduate student researchers during the Spring/Summer of 2015. Interviews with the code owners of the SC projects were conducted before and after redevelopment by an independent researcher. Information about the interview process and questions used for data collection are described Section 4.1.2. Details on the ethics application can be found in Appendix A.

4.1.1 Criteria for a Candidate Redevelopment Project

The software included in the study was selected based on the following criteria:

1. The software should emphasize SC, either in terms of solving a specific physical problem, or as a general purpose SC tool.
2. The software should be small to medium in size, between 1000 to 5000 lines of code.
3. The software should not be too complex or require too much domain knowledge. The students redeveloping the software will have either a computer science, mathematics, or software engineering background. They are comfortable with mathematics, and in some cases numerical methods, but most do not have a strong background in the physical sciences.
4. The software should be of continuing interest to its owner, and there should be future plans to use it.

5. A command line interface is preferred over a GUI, since we are only interested in insight into development of SCS.

6. While the specific programming language does not matter, the source code should be available.

7. Some documentation other than the source code should be available in the form of a user guide, theory manual, developer's guide, graduate thesis, or undergraduate thesis.

8. The owner should be available to answer questions.

Seven SC projects (including the Solar Water Heating System (SWHS) pilot project, see Chapter 3) were selected for redevelopment. All seven projects initially met the criteria described in Section 4.1.1, except for the last criterion, which will be discussed further in Chapter 5. Six of the participants are academic scientists or engineers and their software is developed for their own research needs. One participant is an industrial scientist. While in this case the software itself is not a commercial product, it is used to create a commercial service. In all cases, the participants held a PhD in their domain of interest.

4.1.2 Redevelopment and Data Collection Methods

The seven SCS projects were redeveloped using a Document Driven Design (DDD), as described in Section 2.3. The specific technologies and methods varied depending on the project, but the general process of DDD was applied. For each project, a subset of the following artifacts were produced:

- Requirements document
- Design document
- Code
- Test plan and test results

The implementation of the code can be traced to the requirements and design documents. The software is supported by tools commonly used as part of best practices in software development, such as the use of a concurrent versioning system and a proper build system, which supports generating documentation, executable code and regression tests.

Details about the interviews

To measure the effectiveness of DDD, the attitudes of the software owners were recorded before and after the redevelopment through interviews conducted by an independent researcher: Dr. Diane Kelly, from the Department of Mathematics and Computer Science at the Royal Military College of Canada. The purpose of the interviews was to ascertain the perception the software owner in regards to the importance of software activities that contribute to improved software quality. Two interviews were conducted with each software owner. One interview took place before the software was redeveloped using DDD approaches and a second interview was conducted after redevelopment. Since perceptions can impact the successful adoption of new software development approaches, a qualitative rather than quantitative assessment was used in the form of open-ended exploratory interviews. The exploratory nature of this study also leads to questions of a qualitative nature. Rather than a yes/no survey, interviewees were encouraged to take control of the interview and comment on broad areas suggested by the interviewer. Prompts such as “tell me more”, “give me some examples”, “are there reasons for this?”, and “what else would you do?” were used to encourage the interviewee to provide more detail. The broad areas suggested to the interviewee are listed below. Each interview lasted between 30 and 60 minutes.

Initial Interview Topics and Questions

Information about the scientist:

- Scientists current position/title? degrees?
- Specific scientific area that the software contributes to?
- Scientists contribution to/relationship with the software?
- Length of time the scientist has been involved with this software?
- What formal instruction has the scientist received in relation to the software work?

Information about the software:

- What language is the software written in?
- How large is the software (LOC)?
- Is the software part of an open source consortium? Is it wholly written in-house?
- How large is the development group?
- How large is the user group?
- What is the typical background of a user?
- What is the typical background of a developer?

Areas of discussion:

1. What is the most important software quality(ies) to your work? What is the worst scenario if the software failed?
2. Do you address any of your quality concerns using documentation? Give examples where the documentation helped and how it helped? Give details of what you include in your documentation? Is there any documentation you feel you should produce and do not? Why?
3. Do you use any tools to assist you in your documentation? Do you have any success stories using it? Stories of failures?

4. Do you use tools to help reproduce previous software results? Success stories? Failures? Is reproducibility important to you? (e.g. version control, configuration management)

5. What do you do to build confidence in your code?

   - discussion about testing: do you use any tools to support testing? e.g. unit testing tools, regression testing suites? Profilers? Success or failures?

   - discussion about other verification/validation activities: tool support? What do you find the most effective way to build confidence in your software?

6. Do you use other peoples software? How do you ensure it is trustworthy? What do you find is the greatest difficulty in using other peoples software? Ideally, what would you like to see that would help?

7. Give an overview on how you typically develop software. For example, on a typical day, what activities would you be engaged in? Do you find your time is used efficiently? If not, what would help?

Second Interview Questions

1. What are the most noticeable differences between the previous version of your software and the redeveloped version? What was your reaction to the changes? Have

you had a chance to work with the new version? Any comments? Have you tested the new version to ensure nothing important was changed? Comments?

2. If you feel the new software version benefited from the redevelopment, what do you think drove the greatest benefit? The document driven process used? Someone simply spending time on the software? Something else?

3. Going forward, will you use any of the software development support tools from this project? For instance, if you do not already do so, will you use version control in the future? If not, why not?

4. Going forward, will your approach to documentation of requirements and design change? If not, why not?

5. Going forward, will your approach to verification and validation change? If you do not already, will you use unit testing and regression testing in the future? If not, why not?

6. Will this experience influence how you develop software? Do you see yourself maintaining the same level of documentation, tool support as you go forward? What benefits for you as a software developer/user/other have you gained from this exercise?

4.2 Overview of Selected Case Studies

This section contains a brief description of each selected projected and the results of the first set of interviews. To protect the anonymity of the participants, each project was given a pseudonym. Additionally, identifying details were changed or omitted, and gender neutral plural pronouns were used, even when the pronoun should, strictly speaking, be singular. Due to the free-form nature of the interviews, not all questions were answered. Table 4.1 summarizes the first set of interviews and the characteristics of the original software and tools used. Table 4.2 summarizes the software artifacts produced for each project. The full transcriptions of the first set of interviews can be found in Appendix B.

4.2.1 SWHS

To help determine how to best measure the effectiveness of DDD, a pilot study with Dr. Marilyn Lightstone in Mechanical Engineering at McMaster University was conducted. While the software fit the criteria outlined in Section 4.1.1, the interviews that were conducted differed from the interview process given in Section 4.1.2. For this pilot project, code, an automated testing framework, a Software Requirements Specification (SRS), Module Guide (MG), Verification and Validation (V&V) Plan, and V&V Report

Table 4.1: Overview of Case Studies

	SWHS	**Astro**	**Glass**	**Soil**	**Neuro**	**Acoustic**	**GamePhysics**
Size (LOC)	1000	5000	1300 (back end)	800	1000	200	1000
Language	FORTRAN 77	C	FORTRAN 77 (back end)	Matlab	Matlab	Matlab	C
# Developers	1	2	1	1	1	4	1
Age of Software (years)	5	10	<1	5	5	2.5	5
SE Experience?	No	No	No	Yes	Yes	No	N/A[1]
Programming Experience?	Yes	Yes	Yes	Yes	Yes	Yes	N/A[1]
Background	PhD Mechanical Engineering	PhD Astrophysics	PhD Civil Engineering	PhD Civil Engineering	PhD Neuro-science	PhD Chemical Engineering	N/A[1]
Automated Testing Used?	No	No	No	Yes	No	No	N/A[1]
Documentation	Handwritten notes, journal articles, code comments	User manual, journal articles, code comments	Code comments	User manual, code comments, conference presentation	Code comments	User manual, code comments	User manual, API reference, tutorial documents
Version Control Tool Used?	No	No	No	Yes	Yes	No	Yes
Users	Developer and graduate students	Developers and other domain experts	Developer	Undergraduate students	Developer	Developers	Open source
Important Software Qualities	Correctness	Correctness, usability, reproducibility, backwards compatibility	Not answered	Not answered	Safety of output	Not answered	N/A[1]
Bug Tracking Tool Used?	No	No	No	No	No	No	N/A[1]

[1] No interviews were conducted -see Section 5.1.6

Table 4.2: Software Artifacts Delivered

	SWHS	**Astro**	**Glass**	**Soil**	**Neuro**
Code	Yes	Yes	Yes	Yes	No
SRS	Yes	Yes	Yes	Yes	Yes
MG	Yes	Yes	Yes	Yes	No
MIS	No	Yes	No	Yes	N/A[1]
Testing Plan and Report	Yes	Yes	No	Yes	Yes
Language	Matlab	C	Python	Matlab	N/A[1]

[1] Code was not redeveloped-see Section 5.1.4

were produced.

The software that was redeveloped was mechanical engineering software that modelled the temperature and energy of a solar water heating tank containing Phase Change Material (PCM) over time. The project is small (approximately 1000 lines of code), 4 years old, and was written in FORTRAN 77. The developer is a domain expert with extensive programming experience but no formal SE training. The developer is also a user, alongside several graduate students who are extending the model. The software is used alongside experimental research that is conducted by the same research group.

Prior to redevelopment, no traditional SE documentation existed. The mathematical model was documented and derived in handwritten notes [43], as well as a Masters thesis [57]. No tools were used for version control, or testing, and testing was not automated. Select test cases were compared to analytical solutions and experimental data to determine the correctness of the software.

4.2.2 Astro

Astro is astrophysics software. It is a medium sized project that is about 5000 lines of code written in C. The software is developed in an academic environment by two developers who are domain experts holding PhDs in their field. The study participant is one of the developers, but their primary role is as a user, tester, and designer. They provide user support (bug reporting) and are responsible for writing and maintaining the user manual. The participant has no formal SE training, but has considerable prior programming experience.

The original software is approximately 10 years old, with new versions released every 1-2 years. The direction of the new versions is made in response to scientific questions. No version control tool is used, however manual version control is used with publicly released versions using a consistent numbering convention. Whenever changes are made, regression testing is performed through all documented examples, but the testing is not automated.

The source code is freely available. Many users are international. On average there are 10 users at a time who are also domain experts. The developers work closely with the users to find bugs. When bugs are reported by the users, further testing outside regular test cases is done and a text file is used to record the bugs.

No formal SE documentation exists. User documentation containing examples, a high-level description of theory, and the format of the input and output of the software is available. The mathematical models and symbols used are documented in journal papers. The symbols used in the journal articles are kept consistent between the publications and the code. The code also contains detailed comments.

For the redevelopment project, the code was redeveloped in C. SRS, MG, and Module Interface Specification (MIS) documents were produced.

4.2.3 Glass

Glass is structural mechanics software to calculate the resistance of a glass window to the pressure of an explosion. The project is small, with a back end of approximately 1300 lines of code. The project duration is approximately one year. The back end of the software was written in FORTRAN 77, and the front end was written in Microsoft Visual Basic. There is one developer, who is a domain expert holding a PhD in Civil Engineering, developing in an academic environment. The developer has no formal SE training, and while he has some prior programming experience, this project was his first significant opportunity to program.

At the moment, the developer is the only user of the software, but the software is being written for a client. The software has a Graphical User Interface (GUI) in Microsoft Excel, so future users do not need to know details of the theory or code to use it. The users only need to know how to use Excel and have knowledge of the applicable ASTM (American Society of the International Association for Testing and Materials) standards.

There is no existing documentation, but there are comments in the code. No version control tool is used, however older versions are manually kept in a directory with a specific numbering convention. Testing is done manually by comparing output with hand calculations and standard design charts.

For the redevelopment project, the back end code was redeveloped in Python. SRS and MG documents were produced.

4.2.4 Soil

Soil is geophysics software that performs stability analysis of a slope. The project is small, approximately 800 lines of Matlab code. There is one developer, who adapted Soil from an earlier program in 2010. The developer is a domain expert, holding a PhD in Civil Engineering, and has prior SE training and programming experience.

The software is used as an educational and instructional tool for undergraduate students in a foundational course to demonstrate slope stability issues and the process of assessing and designing stable slopes. The users are undergraduate students in civil engineering and should have sufficient understanding of the underlying problem to use the software.

No formal SE documentation has been written for this software, however there is tutorial documentation for students. This documentation contains examples and high level explanations of the theory. The code is also extensively commented using the Matlab format of comments. Within the code, meaningful variable names are used.

Both a version control tool (Git) and automated testing are used. The Matlab unit testing framework is not used, however test scripts are used for regression testing and the output is checked against the literature.

For the redevelopment project, the code was redeveloped using Matlab. An SRS, MG, MIS, V&V Plan, and V&V Report documents were produced, as well as an automated testing framework.

4.2.5 Neuro

Neuro is neuroscience software that includes a computational model of the brain. This project was started around 2013 and is medium sized (approximately 2000 lines of code) in Matlab. The software is used to produce a commercial service and the code is considered a trade secret. There is only one developer of the software who is also the only user. The developer holds a B.Eng in Electrical Engineering and a PhD in Neuroscience, and has experience with both SE methods and programming.

The company for which the software was produced is in its earlier stages and no documentation has been written. The code owner has stated that while there are some code comments, more extensive commenting and documentation will be needed as the company grows and new developers are hired.

During the early stages of development, Apache Subversion (SVN) was used for version control, but due to time constraints it is no longer used. Early in development, the Matlab debugger was also used for testing with specific test cases along with code reading. At present, testing is done manually by comparing the output of the newer versions of the software with older versions, however perfect reproducibility is impossible since the algorithm used includes randomness.

Since the code is a trade secret, it was necessary to break the criteria listed in Section 4.1.1, and the code was not redeveloped. A modified SRS was produced without the specification of the mathematical model. Testing of the software was performed and an automated testing framework was produced. A Testing Report was written; however, only specific properties of the software were tested, i.e. sanity checks as described in Section 2.3.2.

4.2.6 Acoustic

Acoustic is chemical engineering software that is used to analyze large amounts of acoustic data and produce a waveform that can be related to damage in plastic parts. The software is about 2.5 years old and is part of a project for a company. The study participant is the original developer. Currently there are 4 developers, 3 of which are users. All of the developers have degrees in chemical engineering and some programming experience.

This project was omitted from the study. Further details on why this is the case are given in Chapter 5.

4.2.7 GamePhysics

GamePhysics is a physics library that simulates the movement and collisions of 2-dimensional rigid bodies. The software was written in C. Only a subset of the software of approximately 1000 lines of code, was considered for this project.

The code was redeveloped in C. An SRS, MG, and V&V Plan documents, as well as automatically generated documentation in Doxygen [71], were produced.

This project was omitted from the study. Details on why are given in Chapter 5.

4.3 Participant Responses

This section contains the second interview responses of the five study participants to the code and documentation produced. Some responses have been reworded or paraphrased for clarity. Only the participant feedback in presented here. For discussion of the feedback, see Chapter 5. The full transcripts of the second interviews are found in Appendix C. The Appendix responses are redacted in cases where potentially identifying information was given. Table 4.3 summarizes the participant responses from the second interview. As in

Section 4.2, to preserve the anonymity of the participants, rather than use potentially identifying gender specific pronouns, this section will use the gender neutral plural pronouns, even when the pronoun should, strictly speaking, be singular.

Table 4.3: Second Interview Summary

	SWHS	**Astro**	**Glass**	**Soil**	**Neuro**
Future version control?	No	No	No	Yes	Yes
Future auto testing?	No	No	N/A[1]	Yes	Yes
Maintain SRS?	No	No	Maybe	Maybe	Yes
Maintain MG?	No	Yes	Maybe	Maybe	N/A[2]
Maintain MIS?	No	No	N/A[1]	Maybe	N/A[2]
Testing Plan/Report?	No	No	N/A[1]	Maybe	Yes
Useful aspects of DDD	Documentation of assumptions, variables with units, future changes is useful; SRS could be helpful for a new graduate student	Modules result in more user friendly code; Traceability between modules and requirements is useful	More systematic approach to architecture and organizing information; better organized code	Detailed documentation useful for information sharing on design choices; Documentation and code designed for change; Detailed record of knowledge capital	More thought into design and planning resulting in cleaner design; Code is produced to make testing easier
Drawbacks of DDD	SRS is too long, not necessary for this size problem	DDD will not work in reality, needs upfront requirements; No funding for scientists to write software	Unnecessary layer of abstraction; Too much SE jargon; Use Hierarchy in MG unclear	Requires extra time up front which is not always possible; Unnecessary for small project	Time consuming; Requires extra resources; DDD template needed to be modified to fit FDA standard; Difficult without team of people

[1] Testing was not conducted -see Section 5.1.2
[2] Code was not redeveloped- see Section 5.1.4

4.3.1 SWHS

Since SWHS was a pilot study and not under ethics approval, the second interview with the participant was conducted by Dr. Spencer Smith and Thulasi Jegatheesan. The interview was informal without any predefined questions in mind.

The participant stated that they liked the development procedure, but felt it was too much work and unnecessary for the size of the problem. They said that they liked the explicit statement of the assumptions, as well as the inclusion of units with all variable definitions. Additionally, they agreed that the documentation supported change through the use of modules and traceability.

4.3.2 Astro

The participant stated that the most noticeable of the differences from the redevelopment of the Astro software was the structure of the code. The code was restructured to use the principle of information-hiding. The participant felt that the benefits of the redevelopment process came from a combination of the development approach itself and having an extra set of eyes on the code. The participant felt it was helpful to get an outside perspective on the code from a non-expert in the application domain. The participant stated that they felt that DDD is a powerful approach to software development. They felt that the new code is more user friendly because of the use of information-hiding. Previously they had assumed that any user of Astro would be an expert and would have some familiarity with the code. The newly structured code allows for the user to be eased into the use of the code without having to know the details right away, which is advantageous for both expert and new users. The participant also felt that the MG document would be useful, especially because of the traceability between the requirements and the design.

Although the participant felt that DDD could be a powerful approach, they did not feel that the process is practical, or that is would work in reality. The participant felt that it would be necessary to know the software requirements ahead of time to successfully use

DDD, and that the approach would not work on existing software, only new projects. They also stated that the Canadian funding structure does not support scientists developing software in an academic setting. The funding is for the science itself rather than the software or documentation.

Despite the drawbacks, the participant stated that they would try to incorporate elements of the DDD approach in the future. They said that they would like to keep the traceability between the requirements and modules in the MG up to date. However, they felt that the SRS and MIS documents would be more difficult to maintain. They stated that if a change is to be made to either the SRS or MIS documents, it will be a fundamental change to the theory and would require extensive modification of the SRS and MG. A version control tool was used during the redevelopment process. The participant currently uses manual version control and feels that this is sufficient as new versions are only released every 1 to 2 years.

4.3.3 Glass

The participant felt that the most significant change to the code was the organization, the output format, and the interface. The original Excel interface, which has the potential to break down with new versions, was replaced with Python.

Previously the participant was not using any systematic approach when writing the code and felt that DDD gives a more systematic approach to building the software architecture. They felt that this approach could be useful since the architecture of the original code had to be modified from the first version. They also stated that the SRS provided a systematic way of organizing information. Particularly, the data constraints table in the SRS was found to be useful, however several of the less obvious constraints were not present. The participant also stated that the future maintainability of projects would be better using DDD.

The participant found that the DDD approach added a layer of abstraction and jargon that was unnecessary. They found that most of the detail in the SRS was an explanation of things that the participant "knows well". They also suggested that Use Hierarchy in the MG would be more explanatory if it was restructured as a Venn diagram. They stated that as it was, the mapping of the modules seemed to be backwards. For this project, Pydoc was used to generate some documentation from the code. The participant said this would be more useful if there was an interactive map of the whole system where you could click on specific modules for more details. The participant had also hoped that the interpolation algorithm would be structured in a different way, however after redevelopment, the interpolation algorithm looked the same. They also felt that the code commenting could be more explanatory.

When asked if any changes would be made to future development, the participant responded that for a large project, DDD could be useful since it is a top-down approach that requires the architecture to be thought about upfront. Since the participant was not familiar with how to generate the SRS and MG documents, they said that they would only keep these documents up to date if it did not require a lot of time. A version control tool was also used for the redeveloped project. The participant felt that using a version control tool was unnecessary for a small project with only one developer and felt that their manual version control would suffice.

4.3.4 Soil

The most noticeable changes to the redeveloped software, according to the participant, was the structure of the software. The redeveloped software was divided into significantly more modules based on functionality. While the original code had six modules, the redeveloped code had 12 modules. Additional test cases were also added. Previously there were 5 or 6 system test scripts. After redevelopment, there was automated testing for a wider number of test types, i.e. unit testing was also included for each module.

The participant felt that there were several benefits to the DDD approach. In general, the participant felt that spending initial time upfront is useful and the DDD approach serves as a useful guide that results in more useful output. The documentation that was produced provided a greater amount of detail. They felt that when the design choices are documented explicitly, they are easier to share with a team and the documents keep a record of the design choices for the author as well. The documentation gives a high level detailed record so the knowledge capital that has been built up will not be lost. Producing an SRS forces you to specify the purpose of the program ahead of time, making the developer more accountable. It also provides a record of what the software is supposed to do and provides something to check against. The code produced using this approach is designed for change, as well as the documentation, by using abstraction.

The DDD approach seemed unnecessary for a small project, according to the participant. The extra time upfront for the development process and extra work required is a hurdle. The participant stated that the DDD approach could be useful, especially on larger projects that are shared with other developers, but they felt that it may be difficult to convince collaborators that the process is worth the extra time.

This participant was already using several SE methods and tools prior to redevelopment. The version control tool Git was already in use as well as automated testing. The additional automated unit testing that was added during the redevelopment will be continued and the participant stated that their verification and validation are already performed in a similar way. The participant said that they will try to keep the additional documents produced during the redevelopment up to date and it may be useful in the future if students will be changing the code.

4.3.5 Neuro

The software was not redeveloped for this project, however the participant saw several benefits to using the DDD approach. The participant stated that DDD forces you to put effort into design and planning, which results in a cleaner design that is easier to test. The primary concern of this participant was to obtain Food and Drug Administration (FDA) approval for their software. FDA standards require that requirements and test planning be completed ahead of time and the participant felt that DDD process and documents mapped well to the process and documents required for FDA approval.

The participant believed that the DDD approach would be challenging for a single person to implement and would require a team of people. As part of a start-up, they felt DDD could use up resources and time when the priority would be getting the code to work as quickly as possible.

Since many aspects of the DDD approach overlap with FDA guidelines, the participant stated that the SRS incorporated into the FDA documents will be kept updated to aid future owners of the software system. The participant felt they had a better appreciation of formal methods and testing, and would use the testing methodology that was introduced for future testing. The participant is also planning to continue using a version control tool going forward.

Chapter 5

Discussion

This chapter is divided into three sections. The first section, Section 5.1, describes any divergence from the case study methodology described in Section 4.1. The next section, Section 5.2, analyses common responses by the code owners. The last section, Section 5.3, gives the redevelopment team impressions of the Document Driven Design (DDD) process.

5.1 Divergence from Intended Case Study Process

Due to time constraints and other limitations, which are discussed below, the intended case study methodology was not followed for all of the projects. One of main complications that affected all of the selected projects was the delay in ethics approval. Ethics approval was required to include human participants in this study and needed to be granted to contact any participants. The projects were redeveloped during the spring and summer of 2015, however ethics approval was granted late in the summer. In Section 4.1.1, it was stated that the code owner must be available to answer questions, but given the timing of the project, we were unable to involve them until late in the redevelopment. Had the code owners been engaged from the beginning of the redevelopment, the results of the interviews may have been different. The quality of the documentation and code would have been improved through iteration; unfortunately, the participants had to receive a final version of the code and documentation that did not benefit from their feedback. Furthermore, we were unable to consult with them to obtain missing or misinterpreted information, and we were unable to explain the reasoning behind the development process that was used. Further details about each project can be found below.

5.1.1 Astro

Prior to the second interview, the code owner did not have time to go through the documentation and code in detail. Moreover, due to the minimal time for communication, we were unable to explain some of the Software Engineering (SE) terminology used in the documentation. Some of the participants' feedback showed a misunderstanding of the redeveloped software and the development process. For instance, we were also unable to convey that while the final documents produced by DDD are written to give the impression that a systematic SE process was followed, the development process itself need not be conducted in a linear manner.

5.1.2 Glass

Due to the unexpected absence of the student developer assigned to this project, the complete set of documents were not produced and no testing phase was conducted. A complete Software Requirements Specification (SRS), Module Guide (MG), and redeveloped code were delivered, but no testing documents, Module Interface Specification (MIS) document, or automated testing framework were produced. The code owner found there was too much SE terminology in the documents and stated that the code comments were not sufficiently descriptive. They also found that the Data Constraints table in the SRS was not complete. Increased communication with the code owner would have facilitated a better understanding of the physical problem and the code owner's expectations on our part, and a better understanding of the SE specific terminology on the code owner's part.

5.1.3 Soil

Communication was limited with the code owner of this project, however the code owner was familiar with the DDD process prior to the redevelopment. It is unlikely that the delay in ethics approval affected the results of the interviews.

5.1.4 Neuro

The goal of this project differed from that of the others. The main goal of the code owner was to obtain Food and Drug Administration (FDA) approval for the product produced by their software. The software contained trade secrets, so it was not available for redevelopment using the DDD approach. Instead, the emphasis was on testing specific qualities of the product produced by the Neuro software. Unlike the other projects, the testing was motivated only by the safety of the final product produced by the software, rather than the correctness and reliability of the software. A hazard analysis of Neuro was performed by a separate research team that incorporated the results of the testing. A high level SRS was produced as well; however, the mathematical model was not documented because it was considered a trade secret. The code owner extended several of the documents produced during the redevelopment, since they were required by the FDA. This project continued into Fall 2015, so no miscommunication occurred due to lack of contact with the code owner.

5.1.5 Acoustic

The code owner of this project was not available during redevelopment. We had limited contact with a graduate student rather than the code owner themselves. Since the availability of the code owner was part of the criteria for selection (see Section 4.1.1) and was not satisfied, this project will not be discussed further in this study.

5.1.6 GamePhysics

The redevelopment of this project included a complete SRS, MG, and Verification and Validation (V&V) Plan, as well as redeveloped code and an automated testing framework. We did not receive a response from the code owner to our requests for participation in this project. Interviews were not conducted with this code owner so this project will not be included in this study.

5.2 Common Participant Responses

This section gathers the common responses given by the participants. It also attempts to clarify any misconceptions about the DDD process.

Documentation and Implementation

All of the participants found some aspect of the documentations useful. Each participant found different sections of the SRS useful, but felt much of the information provided in the SRS was unnecessary for their purposes. The SRS documents were however, intentionally "over-specified", as if they were for safety-related software or for knowledge capture. Most of the participants had a more favourable view of the MG. For many Scientific Computing (SC) developers, producing working code that expresses their science is their main priority and the MG provides a roadmap of the design that results in easier code maintenance and extensibility. In terms of the implementation, the code owners for which a new implementation was provided agreed that the modular code was more organized and accessible.

A high level of detail was provided in the documentation for several reasons. Firstly, documentation was produced as if each project was safety-related and aimed to eliminate any ambiguity about the theory behind the mathematical model, the design of the implementation, and when applicable, the test cases carried out. Secondly, the purpose of the documentation was not only for the current developers, but for future developers and users.

Explicit documentation can also help to facilitate conversations about the theory and design, which can lead to better quality documents and code. For example, the SRS document for the Glass project resulted in discussion about the less obvious data constraints that were excluded from the document. Detailed documentation capturing the knowledge built up in the project allows for an easier transition for new members. Lastly, highly detailed documentation was produced as a proof of concept. Previous studies [11] claim that agile methods are more appropriate for SC developers. The documentation was produced to show that it is possible to successfully create detailed documentation for Scientific Computing Software (SCS).

Design Process

Every participant felt that the DDD process would require a larger time commitment, since producing documentation is the focal point of the process. All participants, except for one, felt that any rewards from using DDD for a small project would be outweighed by the resources needed to complete the documentation. Many felt that the amount of documentation was unnecessary for small projects but one participant stated that a larger project with many developers could benefit from such a process. In general, the priority of the participants was to produce publishable scientific results, but all of the participants found some aspect of the DDD beneficial and agreed that there is value in using a systematic approach to designing software.

SC development often requires flexibility. For this reason, DDD does not prescribe to any particular rigid development process. It only suggests that documentation be produced as if a process were followed (see Section 2.3), as recommended in [52]. This idea was not well communicated to the study participants. Unfortunately, some of them received the impression that the requirements must be known ahead of time to apply the DDD process. The documentation can be created after the requirements are more stable, and the structure of the documents are conducive to change if required. Additionally, many of the participants felt that DDD required more time at the beginning of a project. While extra time will be required, it does not necessarily have to be committed at the beginning of the project, and can ultimately save time overall. For example, one participant stated that the entire architecture of their software had changed from a year previous. In this case, the implementation would not have to have been rewritten had the design been considered in advance.

Furthermore, some participants did not see the value in the use of SE tools and practices, such as version control, issue tracking, and automated testing. These tools and practices require minimal effort to incorporate, when compared to the rest of the DDD process and can have a positive impacted on software quality when used correctly. They are often cited as best practices for SCS development [29, 74, 75].

Analysis of Responses and Concluding Remarks

The sample size of this study is very small and produced qualitative results, making it difficult to draw all-encompassing conclusions from the interview responses. A small project size was a prerequisite for inclusion in this study. The inclusion of larger projects with more developers may have produced different responses from the participants. Additionally, journal policies and research funding could have an impact on the use of SE in SC, and promote software qualities that are not always given importance by SC developers. Journal policies requiring the publication of research software could promote understandability. Funding structures that promote long-term development efforts for more sustainable SCS could result in more maintainable SCS and documentation.

For small projects with a few developers, organizing development efforts is simpler, making it easier to produce software of acceptable quality without the use of a systematic development process or software tools to support development. This becomes increasingly difficult for large projects with many developers, as coordinating development efforts on a larger scale is more complicated. In a previous study by Hannay et al. [25], scientists who developed larger projects with a greater number of developers tended to rate SE concepts higher than those who worked on small projects with few developers. The benefits provided by SE methods and tools becomes more evident as project size and number of developers increases. For example, requirements, design, and testing documents allow for easier communication and coordination between developers, and encourages more maintainable software with a longer lifespan. Using version control software allows for easier source management by many developers, and the use of issue tracking software for requirements, features, and bugs provides a collection point of information for all developers involved with the project [29]. The use of SE processes and tools can greatly reduce the effort involved in development, and makes it easier to produce better software.

As previously stated, the goal of most SC developers is to produce scientific results and publish scientific papers. Applying SE to SCS can help scientists produce better science, which could in turn increase the publication output. Reproducibility is one of the cornerstones of science, and one of the foundations of experimental scientific research is the detailed recording of all materials, methods and results in a lab notebook so that experiments can be reproduced and repeated [8]. Similarly, all computational research should be well-documented, modular, and easy to read so that the program and underlying science are evident, even to those who did not write the program [8]. At present, not all journals require the availability of source code upon publication [33, 46]. On its own, sharing SC code does not guarantee reproducibility, however, it is likely that it will help to increase the quality of code written by SC developers [16], since the code must be understandable to their peers [33]. In a case study by Groen et al. [24], the authors noted that the use of SE principles improved the quality of SCS and led to an accompanying increase in the number of publications produced. Journal policies that require the sharing of code could make the application of SE to SCS more widespread, improving the overall software quality and increasing the publication output for a single SCS product. The mandatory sharing of code may result in a more favourable response to the adoption of SE principles.

Additionally, the results of the case study may have differed if scientists from other countries were included. One of the case study participants stated that the Canadian funding structure does not support scientists writing software. Funding agencies from other countries may place a higher emphasis on software development in science and engineering. For example, the National Science Foundation (NSF), which is an United States federal agency, provides funding for scientific software through the SSE and SSI grants [18].

Considerable variation exists in development practices of SC developers. The extent of documentation and use of tools varied between the case study participants. This may be because the development practices of SC developers is frequently dependent on the recommendation of their peers [33]. They are more likely to adopt certain practices if it is endorsed by their colleagues. SC developers are also more likely to adopt tools, techniques, and methods that do not require a change in roles or work practices [59]. In this case study, the majority of the redevelopment effort was spent on documentation and a new implementation. Future case studies attempting to promote SE practices may benefit from focusing on integrating tools and testing techniques, which would not disrupt workflow, to the extent of extensive documentation. All of the participants of this case study were open to the adoption of new practices, as long as drastic changes to their development practices were not required.

Lastly, the results of the case study were dependent on the questions asked by the interviewer. An attempt was made to remove bias and not include leading questions. However, additional questions may give a better picture of the the current practices of the participants and their reception of the DDD process.

The following questions could be added to the first interview to get a better picture of the development practices of the participant. The questions could also guide the redevelopment process by focusing on aspects that the participant found most important.

- Do you use other testing techniques? Eg. mutation testing, pseudo oracles?
- Are there aspects of your documents, design, or testing procedures that you wish to improve?

The following questions could be added to the second interview to clarify their impressions of the DDD process.

- How would you feel about the process if the software was safety-related?
- What if there was a tool to manage and facilitate the documentation?

5.3 Impressions from the Redevelopment Team

Participating in the redevelopment projects gave the redevelopment team, which includes the author, an alternative perspective on the DDD process. Producing an implementation and documentation using DDD gave insight into the benefits and drawbacks of the process and templates.

The development team had SE and programming experience, but in general, the members were not familiar with the scientific domains of the redeveloped projects. For each project, the team had the advantage of consulting the existing implementation. As discussed in Section 5.1, there was limited contact with the code owners during redevelopment, so much of the theory that was used to write the SRS was uncovered from the original implementations, supplemented by journal articles and user manuals. For most projects, the redevelopment was done by a single member, but in some cases, different members were responsible for different parts of the redevelopment for one project.

The redevelopment team agreed that there were several benefits to each redevelopment project. The use of document templates ensured that all necessary and relevant information was recorded such that someone who was not familiar with the theory could understand the project. In one instance, the SRS and implementation were produced by separate team members, but the SRS contained sufficient information for the design and implementation to be produced. One team member stated that the DDD templates present the information in a logical manner and serve as a guide through the software system. The software produced used the design principles separation of concerns and information hiding, resulting in software that was easier to test, understand, maintain, and verify.

Several of the team members pointed out that a primary drawback of the DDD process was the time commitment required to complete it. The documents are laborious to complete. The templates used the typesetting system, LaTeX [3], and while the redevelopment team was familiar with this software, the templates can be overwhelming for someone who is not well-versed in it. Additionally, attempting to maximize the software qualities outlined in Section 2.1 resulted in longer code and a more complex design in some cases. While it may be due to the unfamiliarity of the redevelopment team with the theoretical domains, the redeveloped code likely took longer to write than the original code.

Chapter 6

Conclusions

With the ever increasing use of computational experiments for scientific exploration, the quality of scientific software should be given more importance. Just as scientific theory and experimentation are conducted with careful documentation and rigour, similar practices should be encouraged when developing software for scientific purposes. The goal of this research is to investigate ways of applying well-known and well-studied Software Engineering (SE) processes and technologies to the development of Scientific Computing Software (SCS) in a productive and practical manner. SE practices were successfully applied to the redevelopment of five Scientific Computing (SC) projects:

- Solar Water Heating System (SWHS), thermodynamics software
- Astro, astrophysics software
- Glass, structural analysis software
- Soil, geophysics software

- Neuro, neuroscience software

A summary of the results and main conclusions of the research are given in Section 6.1. Possible avenues of future work are discussed in Section 6.2.

6.1 Summary

Section 6.1.1 addresses the research questions posed in Chapter 1. Section 6.1.2 contains a summary of the results and conclusions of the case studies.

6.1.1 Summary of Research Questions

In this thesis, the SE process, Document Driven Design (DDD) and SE tools are applied to SCS with the purpose of achieving the desirable software qualities defined in Section 2.1. Existing SC projects were redeveloped using the DDD process described in Section 2.3. The DDD process and the software artifacts that resulted from the process were evaluated through an interview process with the code owners of each project. The research questions posed in Section 1.3 are considered below:

1. Can traditional SE methods be applied to SCS?

In each SC project considered, the DDD process and SE tools were applied with some measure of success, based on the opinion of the author and the results of the interviews. Knowing the requirements prior to the redevelopment expedited the process; however, it is not necessary for the requirements to be known a priori to apply SE processes and tools to

SCS. The documentation was produced as if a systematic design process was followed, as recommended by [52]. For each project, a subset of the documents listed in Section 4.1.2 along with modular code and automated testing frameworks were produced successfully.

2. Does DDD help to achieve desirable software qualities?

The use of a systematic development process does not guarantee a better software product; however, if used thoughtfully, it can yield a software product with more predictable qualities. The document templates used in DDD were designed to achieve desirable software qualities. Each document uses the principle of separation of concerns, which allows for easier maintenance and reuse. The document template encourages understandability, consistency, traceability, and completeness, which help to achieve verifiability, reproducibility, and correctness. The structure and implementation of each redeveloped project was designed to promote maintainability, understandability, and reusability through a modularization.

3. How can DDD be refined to best fit the needs of scientists developing SCS?

The largest criticism of the DDD process was the amount of time required to produce the documents. Methods to reduce the time required for DDD will be discussed in Section 6.2. Additionally, SE practices may be more willingly adopted if they do not require major changes in the roles and workflow of SC developers. Future case studies should attempt to incorporate the DDD process into the existing development patterns of the SC developers. The participants should be involved from the start.

6.1.2 Case Study Summary

The main opinions of the case study participants can be summarized as follows:

- Several aspects of the documentation are useful, especially the Software Requirements Specification (SRS) and the Module Guide (MG), but they are too lengthy and detailed for their purposes.
- The modular code was more organized and accessible.
- The DDD process can be valuable for large software projects, but it is too time consuming for small projects, and requires too large of a time commitment, especially at the beginning of a project.

The main conclusions drawn from the case study are summarized as follows:

- Larger projects with more developers may have yielded more positive results for the adoption of SE practices.
- Change in journal policies requiring code submission may result in a greater interest in SE practices by the SC community.
- It is unlikely that SC developers will adopt practices that require significant changes to their development practices. Tools, such as version control, and a focus on new

testing techniques may serve as a better introduction to SE practices than extensive documentation.

- Funding structures that promote long-term development efforts for more sustainable SCS could create more interest in SE practices in the SC community.

6.2 Future Work

As explored in Chapter 5, each study participant found different aspects of the DDD process and documentation useful, but felt that either it required too large of a time investment, or that a different level of detail in the documentation would be more useful. Current work by [69] is attempting to address the perceived drawbacks of DDD by building a software tool, Drasil, which can generate the required software artifacts from a single source that contains all relevant information. This tool would allow SC developers to focus on the science of their problem while still improving software qualities. The software qualities in Section 2.1 did not include the performance of the software, since performance is sometimes sacrificed to maximize other software qualities. In addition to improving the software qualities already discussed, Drasil could potentially improve the performance of the software since understandability, maintainability, and reusability of the code becomes less of a concern if the code can be generated. At present, there exist several document generators, such as Javadoc [2], Doxygen [71], and Sphinx [4], that extract code comments and generate well-formatted web pages or documents. Drasil will extend this functionality to achieve a more thorough knowledge capture, and to generate all software artifacts, not just the code and its Application Program Interface (API).

Another potential solution that could reduce the amount of time and effort required to produce the documents associated with DDD would be providing an online library of completed documents for SC developers to contribute. The structure of the DDD documents are such that reuse is facilitated. The SRS document produced for the SWHS project in Section 3.1 was modified from an SRS documenting the requirements for software modelling a single fuelpin in a nuclear reactor [40]. Many systems that are solving thermodynamics problems will likely reuse much of the SWHS SRS. If a library of such documents existed, the time required to produce the documents would be decreased. Rather than starting from scratch, an SC developer can modify completed documents that have a shared theoretical domain.

Another common criticism of SE processes is the need for upfront software requirements. While we argue that this is not the case with DDD, several of the study participants pointed out that we had the advantage of knowing the requirements of their software prior to beginning redevelopment. Another study is currently under way in partnership with faculty members in Mechanical Engineering at McMaster University. In this project, DDD is being applied to an SC problem in its early stages. No implementation exists as yet and the requirements are not static. In the current study, the intended knowledge transfer did not occur with the study participants. The hope is that involving the code owners from the beginning will facilitate a more complete understanding by the participants of the DDD process and SE tools. This study will also help to determine whether the full software requirements must be known ahead of time to apply DDD to SC development.

Bibliography

[1] Git. https://git-scm.com/. Accessed: 2016-07-19.

[2] Javadoc. http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html. Accessed: 2016-07-19.

[3] LaTeX–a document preparation system. https://www.latex-project.org/. Accessed: 2016-07-19.

[4] Sphinx python documentation generator. http://www.sphinx-doc.org/en/stable/. Accessed: 2016-07-19.

[5] David H. Bailey, Jonathan M. Borwein, and Victoria Stodden. Facilitating reproducibility in scientific computing: Principles and practice, 2014.

[6] Jordan B. Barlow, Justin Giboney, Mark J. Keith, David Wilson, Ryan Schuetzler, and and Anthony Vance Paul Benjamin Lowry. Overview and guidance on agile development in large organizations. *Communications of the Association for Information Systems*, 29(2):25–44, 2011.

[7] Earl T. Barr, Mark Harman, Phil McMinn, and Muzammil Shahbazand Shin Yoo. The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, 41(5):507–525, 2015.

[8] Susan M. Baxter, Steven W. Day, Jacquelyn S. Fetrow, and Stephanie J. Reisinger. Scientific software development is not an oxymoron. *PLoS Comput Biol*, 2(9):e87, 2006.

[9] Alberto Bacchelliand Christian Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 international conference on software engineering*, pages 712–721. IEEE Press, 2013.

[10] Jeffrey C. Carver, Lorin M. Hochstein, Richard P. Kendall, Taiga Nakamura, Marvin V. Zelkowitz, Victor R. Basili, and Douglass E. Post. Observations about software development for high end computing. *CTWatch Quarterly*, 2(4A):33–38, November 2006.

[11] Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. Software development environments for scientific and engineering software: A series of case studies. In *Proceedings of the 29th international conference on Software Engineering, ICSE'07*, Minneapolis, United States, May 2007.

[12] G. Chang, C. B. Roth, C. L. Reyes, O. Pornillos, Y. J. Chen, and A. P. Chen. Retraction. *Science*, 59(314):1875, 2006.

[13] M. R. Collins, Frank J. Vecchio, Robert G. Selby, and Pawan R. Gupta. The failure of an offshore platform. *Concrete International*, 19:28–36, 1997.

[14] Daniel Daniel Hook and Diane Kelly. Testing for trustworthiness in scientific software. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 59–64. IEEE Computer Society, 2009.

[15] Martin D. Davis and Elaine J. Weyuker. Pseudo-oracles for non-testable programs. In *Proceedings of the ACM'81 Conference*, pages 254–257. ACM, 1981.

[16] Steve M. Easterbrook. Open code for open science? *Nature Geoscience*, 7(11):779–781, 2014.

[17] Michael E. Fagan. Design and code inspections to reduce errors in program development. In *Pioneers and Their Contributions to Software Engineering*, pages 301–334. Springer, 2001.

[18] National Science Foundation. Software infrastructure for sustained innovation. http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=504865, 2016. Accessed: 2016-07-04.

[19] The Apache Software Foundation. Apache subversion. *URL: https://subversion.apache.org/*, 2011.

[20] Martin Fowler and Jim Highsmith. The agile manifesto. *Software Development*, 9(8):28–35, 2001.

[21] Marc-Oliver Gewaltig and Robert Cannon. Quality and sustainability of software tools in neuroscience. *Cornell University Library*, pages 1–20, 2012.

[22] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2002.

[23] Matheus Grasselli and Dmitry Pelinovsky. *Numerical Mathematics*. Jones and Bartlett Publishers, United States, 1st edition, 2008.

[24] Derek Groen, Xiaohu Guo, James A. Grogan, Ulf D. Schiller, and James M. Osborne. Software development practices in academia: a case study comparison. *arXiv preprint arXiv:1506.05272*, 2015.

[25] Jo Erskine Hannay, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson. How do scientists develop and use scientific software? In *Proceedings of the 2009 ICSE workshop on Software Engineering for Computational Science and Engineering*, pages 1–8. IEEE Computer Society, 2009.

[26] Les Hatton and Andy Roberts. How accurate is scientific software? *IEEE Trans. Softw. Eng.*, 20(10):785–797, 1994.

[27] Leslie Hatton. Software failures-follies and fallacies. *IEE Review*, 43(2):49–54, 1997.

[28] Dustin Heaton and Jeffrey C. Carver. Claims about the use of software engineering practices in science: A systematic literature review. *Information and Software Technology*, 67:207–219, November 2015.

[29] Michael A. Heroux and James M. Willenbring. Barely sufficient software engineering: 10 practices to improve your CSE software. In *Software Engineering for Computational Science and Engineering, 2009. SECSE'09. ICSE Workshop on*, pages 15–21. IEEE, 2009.

[30] Darrel Ince. The duke university scandal–what can be done? *Significance*, 8(3):113–115, 2011.

[31] Cezar Ionescu and Patrik Jansson. Dependently-typed programming in scientific computing. In *Symposium on Implementation and Application of Functional Languages*, pages 140–156. Springer, 2012.

[32] Stephen C. Johnson. *Lint, a C program checker*. Citeseer, 1977.

[33] Lucas N. Joppa, Greg McInerny, Richard Harper, Lara Salido, Kenji Takeda, Kenton O'Hara, David Gavaghan, Stephen Emmott, et al. Troubling trends in scientific software use. *Science*, 340(6134):814–815, 2013.

[34] Upulee Kanewala and James M. Bieman. Testing scientific software: A systematic literature review. *Information and software technology*, 56(10):1219–1232, 2014.

[35] Diane Kelly. Industrial scientific software: a set of interviews on software development. In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, pages 299–310. IBM Corp., 2013.

[36] Diane Kelly. A software chasm: Software engineering and scientific computing. *IEEE Software*, November/December2007.

[37] Diane Kelly and Rebecca Sanders. Assessing the quality of scientific software. In *First International Workshop on Software Engineering for Computational Science and Engineering*, Leipzig, Germany, May 2008.

[38] Diane Kelly and Rebecca Sanders. The challenge of testing scientific software. *CAST 2008: Beyond the Boundaries*, 2008.

[39] Diane Kelly, Spencer Smith, and Nicholas Meng. Software engineering for scientists. *IEEE Computer*, 13(5):7–11, 2011.

[40] Nirmitha Koothoor. A document drive approach to certifying scientific computing software. Master's thesis, McMaster University, Hamilton, Ontario, Canada, 2013.

[41] Friedrich Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In *Proceedings of CompStat2002*. Springer Verlag, March 2002.

[42] Randall J. LeVeque, Ian M. Mitchell, and Victoria Stodden. Reproducible research for scientific computing: Tools and strategies for changing the culture. *Computing in Science and Engineering*, 14(4):13, 2012.

[43] Marilyn Lightstone. Derivation of tank/pcm model. Personal Notes, 2012.

[44] Anders Lundgren and Upulee Kanewala. Experiences of testing bioinformatics programs for detecting subtle faults. In *Proceedings of the International Workshop on Software Engineering for Science*, pages 16–22. ACM, 2016.

[45] Alan MacCormack, John Rusnak, and Carliss Y. Baldwin. The impact of component modularity on design evolution: Evidence from the software industry. Technical Report 08-038, Harvard Business School, 2007. working paper.

[46] A. Morin, J. Urban, P. D. Adams, I. Foster, A. Sali, D. Baker, and P. Sliz. Shining light into black boxes. *Science*, 336(6078):159–160, 2012.

[47] William L. Oberkampf and Matthew F. Barone. Measures of agreement between computation and experiment: Validation metrics. *Journal of Computational Physics*, 217:5–36, 2006.

[48] William L. Oberkampf, Timothy G. Trucano, and Charles Hirsch. Verification and validation for modeling and simulation in computational science and engineering ap-

plications. In *Foundations for Verification and Validation in the 21st Century Workshop*, 2002.

[49] D. L. Parnas, P. C. Clements, and D. M. Weiss. The modular structure of complex systems. In *ICSE '84: Proceedings of the 7th international conference on Software engineering*, pages 408–417, Piscataway, NJ, USA, 1984. IEEE Press.

[50] David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM, vol. 15, no. 2, pp. 1053-1058*, December 1972.

[51] David L. Parnas. On the design and development of program families. *IEEE Transactions on Software Engineering*, SE-2(1), March 1976.

[52] David L. Parnas and P.C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, 12(2):251–257, February 1986.

[53] Marian Petre and Greg Wilson. Code review for and by scientists: preliminary findings. 2014.

[54] Patrick J. Roache. Quantification of uncertainty in computational fluid dynamics. *Annual review of fluid Mechanics*, 29(1):123–160, 1997.

[55] Christopher J. Roy. Practical software engineering strategies for scientific computing. In *Proceedings of the 19th AIAA Computational Fluid Dynamics Conference*, pages 1473–1485, 2009.

[56] Geir Kjetil Sandve, Anton Nekrutenko, James Taylor, and Eivind Hovig. Ten simple rules for reproducible computational research. *PLOS Computational Biology*, 9(10), October 2013.

[57] Padideh Sarafraz. Thermal optimization of flat plate PCM capsules in natural convection solar water heating systems. Master's thesis, McMaster University Hamilton, 2013.

[58] Judith Segal. When software engineers met research scientists: a case study. *Empirical Software Engineering*, 10(4):517–536, 2005.

[59] Judith Segal. Some problems of professional end user developers. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*, pages 111–118. IEEE, 2007.

[60] Judith Segal. Models of scientific software development. In *Proceedings of the 2008 International Workshop on Software Engineering for Computational Science and Engineering*, 2008.

[61] Judith Segal. Software development cultures and cooperation problems: A field study of the early stages of development of software for a scientific community. *Computer Supported Cooperative Work (CSCW)*, 18(5-6):581–606, 2009.

[62] Judith Segal and Chris Morris. Developing scientific software. *IEEE Software*, 25(4), July/August 2008.

[63] S. Smith and W. Yu. A document driven methodology for developing a high quality parallel mesh generation toolbox. *Advances in Engineering Software*, 40(11):1155–1167, 2009.

[64] Spencer Smith, Zheng Zeng, and Jacques Carette. State of practice for seismology software. C.A.S. Report Series, 2015.

[65] W. Spencer Smith. A rational document driven design process for scientific computing software. In Jeffrey C. Carver, editor, *Software Engineering for Science*, chapter Section I – Examples of the Application of Traditional Software Engineering Practices to Science. Submitted 2016. 30 pp.

[66] W. Spencer Smith and Nirmitha Koothoor. A document-driven method for certifying scientific computing software for use in nuclear safety analysis. (in press), 2016.

[67] W. Spencer Smith and Lei Lai. A new requirements template for scientific computing. In J. Ralyté, P. Ågerfalk, and N. Kraiem, editors, *Proceedings of the First International Workshop on Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes, SREP'05*, Paris, France, August 2005. In conjunction with 13th IEEE International Requirements Engineering Conference.

[68] W. Spencer Smith, Lei Lai, and Ridha Khedri. Requirements analysis for engineering computation: A systematic approach for improving software reliability. *Reliable Computing, Special Issue on Reliable Engineering Computation*, 13, 2007.

[69] Dan Szymczak, Spencer Smith, and Jacques Carette. A knowledge-based approach to scientific software development: position paper. In *Proceedings of the International Workshop on Software Engineering for Science*, pages 23–26. ACM, 2016.

[70] Christoph W. Ueberhuber. *Numerical Computation 1: Methods, Software, and Analysis*. Springer, United States, 4th edition, 1997.

[71] Dimitri van Heesch. Doxygen: Source code documentation generator tool. *URL: http://www. doxygen. org*, 2008.

[72] Hans van Vliet. *Software Engineering: Principles and Practice*. John Wiley and Sons, United States, 3rd edition, 2008.

[73] Moshe Y. Vardi. Science has only two legs. *Communications of the ACM*, 53(9), 2010.

[74] Greg Wilson, D. A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H. D. Haddock, Kathryn D. Huff, Ian M. Mitchell, Mark D. Plumblet, Ben Waugh, Ethan P. White, and Paul Wilson. Best practices for scientific computing. *CoRR*, abs/1210.0530, 2013.

[75] Gregory V. Wilson. Where's the real bottleneck in scientific computing? *American Scientist*, 94(1):5, 2006.

Appendix A

Ethics Application

McMaster University

McMaster University Research Ethics Board (MREB)

FACULTY/GRADUATE/UNDERGRADUATE/STAFF

APPLICATION TO INVOLVE HUMAN PARTICIPANTS IN RESEARCH

[Behavioural / Non-Medical]

Date: May 22, 2015	**Application Status: New: [x] Change Request: [ ]**	**Protocol #:**

Helpful Hints Mouse over bold blue hypertext links for help with completing this form.

- **Use the most recent version of this form.**
- **Refer to the McMaster University < Research Ethics Guidelines and Researcher's Handbook >, prior to completing and submitting this application.**
- **For <help> with completing this form or the ethics review process, contact the Ethics Secretariat at ext. 23142, or 26117 or ethicsoffice@mcmaster.ca**
- **To change a previously cleared protocol, please submit the "< Change Request >" form.**

PLEASE SUBMIT YOUR APPLICATION PLUS SUPPORTING DOCUMENTS (scanned PDF signature) BY E-MAIL
You can also send the signed signature page to: **Ethics Secretariat, Research Office for Administration, Development and Support (ROADS), Room 305 Gilmour Hall, ext. 23142**, ethicsoffice@mcmaster.ca.

SECTION A – GENERAL INFORMATION

1. Study Titles: (Insert in space below)

Title: Impact of Document Driven Design in Scientific Computing
1a: **Grant Title:** (*Required for funded research. Click this < **link** > to determine your "grant title").*

2. **Investigator Information:** **This form is not to be completed by < Faculty of Health Science researchers > .**

***Faculty and staff information should be inserted above the black bar in this table.**
Student researcher and faculty supervisor information should be inserted below the black bar in the table below.

	Full Name	**Department** *& or name of university if different from McMaster*	**Telephone Number(s) *& Extension(s)***	**E-mail Address** *(Address you regularly use)*
Principal Investigator*	Spencer Smith	Computing and Software	905-525-9140 X27929	smiths@mcmaster.ca
Co-Investigator(s) *(Insert additional rows as required.)*	Diane Kelly	Department of Mathematics and Computer Science, Royal Military College	613-5416 X6171	kelly-d@rmc.ca
Student Investigator(s)*	Thulasi Jegatheesan	Computing and Software		t.jegatheesan@gmail.com
Faculty Supervisor(s)*	Ned Nedialkov	Computing and Software		nedialk@mcmaster.ca

3. **Study Timelines:** *(Contact the Ethics Secretariat at X 23142 or ethicsoffice@mcmaster.ca for urgent requests.)*
(a) What is the date you plan to begin recruiting participants or obtain their permission to review their private documents (Provide a specific date)?

Monday, June 15, 2015

(b) What is the estimated last date for data collection with human participants?

Wednesday, September 30, 2015

4. **Location of Research**: List the location(s) where research will be conducted. Move your mouse over this **< Helpful Hint >** for more information on foreign country or school board reviews and contact the Ethics Office at X 23142 or 26117 for information on possible additional requirements:

(a) McMaster University	**[x]**	
(b) Community	**[x]** Specify Site(s)	Royal Military College, Kingston, ON
(c) Hospital	**[ ]** Specify Site(s)	
(d) Outside of Canada	**[ ]** Specify Site(s)	
(e) School Boards	**[ ]** Specify Site(s)	
(f) Other	**[ ]** Specify Site(s)	

5. **Other Research Ethics Board Clearance**
(a) Are researchers from outside McMaster also conducting this research? If **yes**, please provide their information in Section 2 above. **[x] Yes [ ] No**

(b) Has any other institutional Research Ethics Board already cleared this project? **[ ] Yes [x] No**

(c) If **Yes** to (5b), complete this application and provide a copy of the ethics clearance certificate /approval letter.

(d) Please provide the following information:

Title of the project cleared elsewhere:
Name of the other institution:
Name of the other board:
Date of the other ethics review board's decision:
Contact name & phone number for the other board:

(e) Will any other Research Ethics Board(s) or equivalent be asked for clearance? **[x] Yes [x] No**
If yes, please provide the name and location of board(s).

Research Ethics Board, Royal Military College, Kingston, Ontario

GENERAL INSTRUCTIONS AND HELPFUL TIPS (Please read first):
*Please be as **clear** and **concise** as possible and **avoid technical jargon**. Keep in mind that your protocol could be read by reviewers who may not be specialists in your field. Feel free to use headings, bolding and bullets to organize your information. Content boxes on this application expand.*

6. **Research Involving Canadian Aboriginal Peoples** i.e., First Nations, Inuit and Métis (Check all that apply)
(a) Will the research be conducted on Canadian Aboriginal lands? **[ ] Yes [x] No**

(b) Will recruitment criteria include Canadian Aboriginal identity as either a factor for the entire study or for a subgroup in the study? **[ ] Yes [x] No**

(c) Will the research seek input from participants regarding a Canadian Aboriginal community's cultural heritage, artifacts, traditional knowledge or unique characteristics? **[ ] Yes [x] No**

(d) Will research in which Canadian Aboriginal identity or membership in an Aboriginal community be used as a variable for the purpose of analysis of the research data? **[ ] Yes [x] No**

(e) Will interpretation of research results refer to Canadian Aboriginal communities, peoples, language, history or culture? **[ ] Yes [x] No**

If "Yes" was selected for any questions 6.a-6.e above, please note that the TCPS (Chapter 9) requires that researchers shall offer the option of engagement with Canadian Aboriginal communities involved in the research. http://www.pre.ethics.gc.ca/eng/policy-politique/initiatives/tcps2-eptc2/chapter9-chapitre9/. For advice regarding TCPS guidelines for conducting research with Canadian Aboriginal peoples, please contact Karen Szala-Meneok at X 26117 or szalak@mcmaster.ca

(f) Please describe the nature and extent of your engagement with the Aboriginal community(s) being researched. The nature of community engagement should be appropriate to the unique characteristics of the community(s) and the research. The extent of community engagement should be determined jointly by the researchers and the relevant communities. Include any information/advice received from or about the Aboriginal community under study. ***The TCPS notes; "although researchers shall offer the option of engagement, a community may choose to engage nominally or not at all, despite being willing to allow the research to proceed". If conducted research with several Aboriginal communities or sub-groups, please use headings to organize your information.***
ATTACHMENTS: Provide copies of all documents that indicate how community engagement has been or will be established (e.g., letters of support), where appropriate.

N/A

(g) Has or will a research agreement be created between the researcher and the Aboriginal community? **[ ] Yes [x] No**

If **Yes**, please provide details about the agreement below (e.g., written or verbal agreement etc.).
ATTACHMENTS: Submit a copy of any written research agreements, if applicable. See the MREB website for a sample customizable research agreement https://reo.mcmaster.ca/educational-resources or visit the CIHR website **http://www.cihr-irsc.gc.ca/e/29134.html**

N/A

(h) Are you are seeking a waiver of the community engagement requirement? (A waiver may be granted if the REB is satisfied that, Aboriginal participants will not be identified with a community or that the welfare of relevant communities will not be affected by the research.) **[ ] Yes [x] No**

If **yes,** please provide the rationale for this waiver request in the space below.

N/A

7. **Level of the Project** (Check all that apply)

[x] Faculty Research **[ ]** Post-Doctoral	**[ ]** Ph.D. **[ ]** Staff/Administration
[ ] Master's (Major Research Paper - MRP)	**[x]** Master's (Thesis)
[ ] Undergraduate (Honour's Thesis)	**[ ]** Undergraduate (Independent Research)
[ ] Other (specify)	

8. **Funding of the Project**
(a) Is this project currently being funded? **[ ] Yes [x] No**
(b) If **No**, is funding being sought? **[x] Yes [ ] No**
(c) Period of Funding: **From: [06/01/2015] To: [11/30/2015]**
(mm/dd/yyyy) (mm/dd/yyyy)

(d) Funding agency (funded or applied to) & agency number (i.e., number assigned by agency), if applicable.
Click this < link > to determine your "*agency number*". (This is not your PIN number).

[ ] CIHR & agency #	**[x]** NSERC & agency #
[ ] SSHRC & agency #	**[ ]** ARB
[ ] Health Canada & agency #	**[ ]** CFI & agency #
[ ] Canada Graduate Scholarship & Agency #	**[ ]** Post Graduate Scholarship & Agency #
[ ] USRA	**[ ]** Other agency & # (*Specify*)

(e): Are you requesting ethics clearance for a research project that was not originally designed to collect data from human participants or their records (i.e., your research project originally did not involve collecting data from humans or their records) but you now intend to do so? **[ ] Yes [x] No**

9. **Conflicts of Interest**

(a) Do any researchers conducting this study, have multiple roles with potential participants (e.g., acting as both researcher and as a therapist, health care provider, family member, caregiver, teacher, advisor, consultant, supervisor, student/student peer, or employer/employee or other dual role) that may create real, potential, or perceived conflicts, undue influences, power imbalances or coercion, that could affect relationships with others and affect decision-making processes such as consent to participate? **[ ] Yes [x] No**

(i) If **yes**, please describe the multiple roles between the researcher(s) and any participants.

N/A

(ii) Describe how any conflicts of interest identified above will be avoided, minimized or managed.

N/A

(b) Will the researcher(s), members of the research team, and/or their partners or immediate family members:

(i) receive any personal benefits (for example a financial benefit such as remuneration, intellectual property rights, rights of employment, consultancies, board membership, share ownership, stock options etc.) as a result of or being connected to this study? **[ ] Yes [x] No**

(ii) If **yes**, please describe the benefits below. (Do not include conference and travel expense coverage, possible academic promotion, or other benefits which are integral to the conduct of research generally).

(c) Describe any restrictions regarding access to or disclosure of information (during or at the end of the study) that the sponsor has placed on the investigator(s), if applicable.

N/A

SECTION B – SUMMARY OF THE PROPOSED RESEARCH

10. **Rationale**

For the proposed research, please describe the *background* and the *purpose* concisely and in lay terms, as well as any overarching research questions or hypotheses to be examined.
Please do not cut and paste full sections from your research proposal.

The goal of this project is to assess whether a document driven design process can effectively be used to improve the reliability, maintainability, reusability and verifiability of scientific computing software. Developers of scientific software typically do not have a background in software development. Our hypothesis is that modern ideas from software engineering can be used to improve software quality.

11. **Participants**

Please use the space below to describe the:
(a) approximate number of participants required for this study
(b) salient participant characteristics (e.g., age, gender, location, affiliation, etc.)
If researching several sub-populations, use headings to organize details for items (a) and (b).

a) The number of participants for this study will be about 6.
b) The salient characteristic of a participant is that they need to have developed scientific computing code.

12. **Recruitment**
Please describe in the space below:
(a) how each type of participant will be recruited,
(b) who will recruit each type of participant,
(c) relationships (if any) between the investigator(s) and participant(s) (e.g. instructor-student; manager-employee, family member, student peers, fellow club members, no relationship etc.),
(d) permission you have or plan to obtain, for your mode of recruitment for each type of participant, if applicable.
If researching several sub-populations, use headings to organize details for items (a) – (d). Click "Tips and Samples" to find the "How to Unpack the Recruitment Details" worksheet and other samples.
ATTACHMENTS: Provide copies of all recruitment posters, advertisements letters, flyers, and/or email scripts etc. and label these as appendices (e.g., Appendix A or 1).

a) The participants will be recruited over e-mail. Potential participants will be selected based on whether they have a scientific computer program of which they are the "owner." That is, we are looking for people that have developed a piece of software that they are currently using for their work. The recruitment script is provided in Appendix A.
b) The participants will be recruited by the principal investigator (Spencer Smith).
c) There is no relationship between the investigators and the participants. The participants will come from academia and industry. They would best be characterized as colleagues, or peers, of the investigators.
d) There are no plans to get permission for our mode of recruitment.

13. **Methods**
Describe sequentially, and in detail all data collection procedures in which the research participants will be involved (e.g., paper and pencil tasks, interviews, focus groups, lab experiments, participant observation, surveys, physical assessments etc. —*this is not an exhaustive list*). Include information about who will conduct the research, how long it will take, where data collection will take place, and the ways in which data will be collected (e.g., computer responses, handwritten notes, audio/video/photo recordings etc**.).**
If your research will be conducted with several sub-populations or progress in successive phases; use sub-headings to organize your description of methodological techniques.
ATTACHMENTS: Provide copies of all questionnaires, interview questions, test or data collection instruments etc. Label supporting documents as appendices (e.g., Appendix A or 1) and submit them as separate documents - not pasted into this application.
Click "Tips and Samples" to find the "How to Unpack the Methods" worksheet and other samples.

The selected scientific software from each participant will be redeveloped using software engineering methods, principles and tools. To measure the effectiveness of the approaches used, there will be two interviews with each participant. The interview questions are summarized in Appendix B. The purpose of the interviews is to ascertain the perception the software owner on the importance of software activities intended to improve software quality. One interview will take place before the software is redeveloped. The second interview will be conducted after the redevelopment.

The first interview will take place in June or early July, 2015. Following this, an undergraduate or graduate student will redevelop the software using a document driven process. The student will be working under the supervision of Smith. Their role will be as software professionals, not as investigators for this study. The second interview will take place when the redevelopment is complete. The target date for this second interview is before the end of September, 2015. The interviews will be conducted by Kelly, either over the phone or via Skype. Kelly will capture each participant's responses using an audio recording, which she will later transcribe. Each interview is expected to last between 30 minutes and one hour.

In addition to the two interviews, each participant will be asked to discuss the redevelopment of their code. During these sessions, we will ask question to help with the redevelopment and we will communicate with the participant to inform them of our progress with the new documentation. The time commitment for answering questions and reviewing documentation will be 3 to 6 meetings of about 0.5 hours each. To protect the time of the participants, the time commitment for discussion will be capped at a 3 hour maximum. Therefore, the total time commitment for the participants over the duration of the project will not exceed 5 hours (two 1 hour interviews and six 0.5 hour discussion sessions.) Depending on the preferences of the participant, the discussion sessions will be held in-person, over the telephone, over Skype, or over e-mail.

With respect to the ownership of the original code and the redeveloped code, the license will stay the same as for the original code. That is, if the original code is under an open source license, this will continue to be the case and the redeveloped code will automatically come under the same open source license. If the original code is proprietary, this same status will be maintained in the new code. If the code owner requires a Non Disclosure Agreement (NDA), then this agreement will be respected by the researchers. The participant is considered as the owner of the original and the redeveloped code. The researchers will delete the original and redeveloped code from their systems at the conclusion of the project (at most two years from the start date), unless the original license permits keeping the code. Similarly the researchers will not share the code, or use it for any purposes other than for this project, unless the original license permits such use.

14**. Secondary Use of Identifiable Data** *(e.g. the use of personally identifiable data of participants contained in records that have been collected for a purpose other than your current research project)*:

(a) Do you plan on using identifiable data of participants in your research for which the original purpose that data was collected is different than the purpose of your current research project? **[ ] Yes [x] No**

If **yes**, please answer the next set of questions:

(b) Do you plan to link this identifiable data to other data sets? **[ ] Yes [x] No**
If **yes,** please describe in the space below:

(c) What type of identifiable data from this data set are you planning to access and use?
[ ] Student records (please specify in the space below)
[ ] Health records/clinic/office files (please specify in the space below)
[ ] Other personal records (please specify in the space below)

N/A

(d) What personally identifiable data (e.g., name, student number, telephone number, date of birth etc.) from this data set do you plan on using in your research? Please explain why you need to collect this identifiable data and justify why each item is required to conduct your research.

The participant's names and contact information will be known to the investigators. This is necessary for the interviews. However, this information will not be disseminated outside of the investigators.

(e) Describe the details of any agreement you have, or will have, in place with the owner of this data to allow you to use this data for your research. ***ATTACHMENTS*: *Submit a copy of any data access agreements.***

N/A

(f) When participants first contributed their data to this data set, were there any known preferences expressed by participants at that time about how their information would be used in the future? **[ ] Yes [x] No**
If **yes**, please explain in the space below.

N/A

(g) What is the likelihood of adverse effects happening to the participants to whom this secondary use of data relates? Please explain.

N/A

(h) Will participants whose information is stored in this data set (which you plan to use for secondary purposes) consent to your use of this data? **[ ] Yes [x] No**
Please explain in the space below.

N/A

15. **Research Database**

Does your research involve the creation and/or modification of a research database (databank) containing human participant information? A research database is a collection of data maintained for use in ***future*** research. The human participant information stored in the research database can be identifiable or anonymous.
[ ] Yes [x] No
If "Yes" was answered to the above question, you will need to fill out and submit MREB's "Supplementary Form for Creating or Modifying a Research Database Containing Human Participant Information" along with this application.
NOTE: If you intend to collect or store personally-identifying health information, now or at a later stage in your research, your protocol must be cleared by Hamilton Integrated Research Ethics Board (HiREB) rather than MREB. For further advice contact MREB at x 23142 or X 26117 or HIREB x 905 521-2100 X 44574.

16. **Experience**
What is your experience with this kind of research? Include information on the experience of all **individual**(s) who will have contact with the research participants or their data. ***For example, you could mention your familiarity with the proposed methods, the study population(s) and/or the research topic.***

The principal investigator (Smith) has expertise on software engineering methods applied to scientific computing software, since this has been the focus of his research since the year 2000. Smith has conducted two studies at McMaster that required ethics approval: one surveying the attitudes of scientific software developers and one where student participants assessed the effectiveness of a computer game for teaching computing.

The co-investigator (Diane Kelly) also has experience with scientific computing and software engineering. Kelly has authored several papers where the developers of scientific computing software were interviewed. For instance, interviews were conducted by Kelly for "Industrial Scientific Software: A Set of Interviews on Software Development" in Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research, pp 299—310, 2013.

17. **Compensation**

		Yes	**No**
(a) Will participants receive compensation for participation?		**[ ]**	**[x]**
	Financial	**[ ]**	**[x]**
	Other (specify)	**[ ]**	**[x]**

(b) If yes was answered for any of the above choices, please provide details. **See < Helpful Hints > for funded research projects.**

N/A

(c) If participants choose to withdraw, how will you deal with their compensation?

N/A

SECTION C – DESCRIPTION OF THE RISKS AND BENEFITS OF THE PROPOSED RESEARCH

18. **Possible Risks**
(a) Indicate if the participants might experience any of the following risks:

i.) Physical risk (including any bodily contact or administration of any substance)?	**[ ] Yes [x] No**
ii.) Psychological risks (including feeling demeaned, embarrassed worried or upset)?	**[x] Yes [ ] No**
iii.) Social risks (including possible loss of status, privacy and / or reputation as well as economic risks)?	**[x] Yes [ ] No**
iv.) Are any possible risks to participants greater than those the participants might encounter in their everyday life?	**[ ] Yes [x] No**

(b) If you checked **yes** for any of questions **i – iv** above, please describe the risk(s) in the space below.

A participant may feel embarrassed if the redeveloped version of their software is superior to the original. Most scientific software developers will recognize that they are not expected to also be experts in software engineering, but there is the chance that someone will feel that a weakness in their background has been exposed.

The participants also face the social risk that their reputation might be damaged if someone deduces their identity and concludes that their code is not well-written.

(c) Management of Risks: Describe how each of the risks identified above will be managed or minimized. Please, include an explanation regarding why alternative approaches cannot be used.

During the interviews, the participants will be reminded of the value of what they are providing. They will be no criticism or judgment of their current approach; we will emphasize that our purpose is to collect their opinions. Although the investigators (Kelly, Smith, Jegatheesan and Nedialkov) will know the identity of the participants, this information will not be shared outside of this group. Any publications will refer to the participants using generic terminology. That is, rather than say the developer of software "ABC" for simulating the production of thin polymer films, we would say the developer of software for simulating polymer processing.

To deal with the social risk, the following text has been added to the Letter of Information/Consent (Appendix C):

"There is also the social risk that your reputation might be damaged if someone deduces your identity and concludes that your code Is not well written. Given our efforts to keep you anonymous, it is unlikely that you will be identified by anyone. In the unlikely event that you are identified, there will be no basis for criticism, since the new documentation that is being created is not currently used by other scientific software developers. The approach used to redevelop the code may turn out to be a recommended approach in the future, but this is not currently the case."

(d) Deception: Is there any deception involved in this research? **[ ] Yes [x] No**

i.) If deception is to be used in your methods, describe the details of the deception (including what information will be withheld from participants) and justify the use of deception.

N/A

ii.) Please describe when participants will be given an explanation about why deception was used and how they will be debriefed about the study (for example, a more complete description of the purpose of the research).
***ATTACHMENTS*: Please provide a copy of the written debriefing form or script, if applicable.**

N/A

19. **Possible Benefits**
Discuss any potential benefits to the participants and or scientific community/society that justify involvement of participants in this study. (***Please note: benefits should not be confused with compensation or reimbursement for taking part in the study***).

The potential benefit for the participants is that the redeveloped software may be of higher quality than their original software. Several months of effort will be spent on improving their software. If they like the improvements, then they may decide to use the new software and/or the associated documentation going forward. As for the original code, the ownership of the redeveloped code belongs to the participant.

SECTION D – THE INFORMED CONSENT PROCESS

20. **The Consent Process**
(a) Please describe how consent will be documented. Provide a copy of the Letter of Information / Consent Form (if applicable). If a written consent form will not be used to document consent, please explain why and describe the alternative means that will be used. While oral consent may be acceptable in certain circumstances, it may still be appropriate to provide participants with a Letter of Information to participants about the study.
Click "Tips and Samples" for the McMaster REB recommended sample "Letter of Information / Consent Form", to be written at the appropriate reading level. The "*Guide to Converting Documents into Plain Language" is also found under *"Tips and Samples".
***ATTACHMENTS*: Provide a copy of the Letter of Information and Consent form(s) or oral or telephone script(s) to be used in the consent process for each of your study populations, where applicable.**

Consent will be documented by each participant signing the letter of information/consent shown in Appendix C.

(b): Please describe the process the investigator(s) will use to obtain informed consent, including who will be obtaining informed consent. Describe plans for on-going consent, if applicable.

The principal investigator (Smith) will ask each participant to provide informed consent. Consent will be documented by the participant electronically signing the consent form. In addition, the participant will be asked for verbal consent at every point of contact. That is, consent will be requested at the start of every interview and every discussion session. This ongoing consent will be documented with a table including dates and tickboxes for consent. This table is included at the bottom of the original consent form, as shown in Appendix C (Letter of Information/Consent).

21. **Consent by an authorized person**
If participants are minors or for other reasons are not competent to consent, describe the proposed alternate consent process. ***ATTACHMENTS*: Attach the Letter of Information and Consent form(s) to be provided to the person(s) providing the alternate consent. Click "Tips and Samples" to find samples.**

N/A

22. **Alternatives to prior individual consent**

If obtaining written or oral documentation of an individual participant's consent prior to start of the research project is not appropriate for this research, please explain and provide details for a proposed alternative consent process. ***ATTACHMENTS****: Please provide any Letters of Information and or Consent Forms.*

N/A

23. **Providing participants with study results**
How will participants be able to learn about the study results (e.g., mailed/emailed brief summary of results in plain language; posting on website or other appropriate means for this population)?

The participants will learn about the study results through the Masters thesis of Jegatheesan. The thesis should be complete by January 2016. If desired, a copy will be sent to them when it is complete.

24. **Participant withdrawal**
a) Describe how the participants will be informed of their right to withdraw from the project. Describe the procedures which will be followed to allow the participants to exercise this right.

The letter of information/consent will explicitly identify their right to withdraw. All that would be required to withdraw would be an e-mail to the principal investigator (Smith).

b) Indicate what will be done with the participant's data and any consequences which withdrawal might have on the participant, including any effect that withdrawal may have on the participant's compensation or continuation of services (if applicable).

If a participant withdraws, all of their data will be deleted, if they so wish. There is no compensation for participation. However, there is a benefit in terms of potentially improved software. If participant wishes, we will complete the redevelopment of their software.

c) If the participants will not have the right to withdraw from the research, please explain.

N/A

25**. SECTION E – CONFIDENTIALITY & ANONYMITY**

Confidentiality concerns the protection, privacy and security of research data. Consult the Data Security Checklist at http://reo.mcmaster.ca/educational-resources for best practices to secure electronic and hard copy versions of data and study documents.

(a) Will the data you collect be kept protected, private and secure from non-research team members?
[x] Yes [ ] No

If **No**, then explain why not, and describe what steps you be put in place to advise participants that data will not be kept protected, private and secure from non-research team members.

N/A

(b) Describe the procedures to be used to ensure that the data you collect in your research will be kept protected, private, and secure from non-research team members. In your description, explain who will have access to the data and what data security measures will be put in place during data transfer and data storage.

The summary of the interview comments will be produced by Kelly. This document will only be shared by Kelly, Smith, Jegatheesan and Nedialkov. The file will be encrypted and password protected. If e-mailed between participants a secure e-mail server will be used. Although the interview comments are not shared with them, the students redeveloping the software will know that the software they are working on is part of a study.

The confidentiality of the original and redeveloped code will also be protected, if required by the original software license. The researchers and student developers will keep the code on secure servers. If necessary to move the code between servers, this will be done via secure ftp, or by using a physical USB key. The student developers will be informed of the license model used by the original software and all steps required by that license will be followed.

(c) Will the research data be kept indefinitely or will it be deleted after a certain time period? Please explain. In your answer, describe why you plan to keep data indefinitely or not. If deleting data after a certain time period, explain why you chose the time period you did. Describe how participants will be informed whether their data will be deleted or not.

Research data, including any audio recordings of interviews, will be destroyed within 2 years after the research has been completed. There is no need to keep this data indefinitely. A time period of 2 year was chosen to ensure that the data will be available for thesis writing and in case the investigators need to make any changes to the resulting research paper. Participants will be informed that their data will be deleted after a period of 2 year upon completion of research through the letters of informed consent (Appendix C).

If required by the software license, the original and redeveloped code will also be deleted by the end of the 2 year duration of the project.

Anonymity concerns whether participant identities are made known or not. The anonymity promised to participants can be different during different stages of research (i.e., during recruitment, during data collection, during data storage, and during the dissemination of research findings).

(d) Describe the extent to which participant identities will be made known in each of the following activities: during recruitment, during data collection, during data storage, and during the dissemination of research findings. In your description, explain what steps or procedures you plan to put in place to keep participant identities unknown in each of those activities.

Recruitment
• The participants will be known to the investigators.
Data Collection
• The participants will be known to the investigators.
Data Storage
• The participants will be known to the investigators.
Dissemination
• The participants will be anonymous. They will not be identified by name or institution in any publications. Their work will be generalized so that it cannot be identified as belonging to them, although their identity might possibly be deduced by someone reading the publications and combining the generic facts. This possibility is explicitly presented to the participants in the Letter of Information/Consent (Appendix C).

SECTION F -- MONITORING ONGOING RESEARCH

26. **Adverse Events, Change Requests and Annual Renewal/Project Status Report**

a) **Adverse events** (Unanticipated negative consequences or results affecting participants) must be reported by faculty researcher or supervisor to the REB Secretariat (Ethics Office – Ext. 23142) and the MREB Chair, as soon as possible and in any event, no more than 3 days after they occur. See: https://reo.mcmaster.ca/policies/copy_of_guidelines#12-0-adverse-events

b) **Changes to cleared research**: To obtain clearance for a change to a protocol that has already received ethics clearance, please complete the "**< Change Request >**" form available on the MREB website or by clicking this link. Proposed changes may not begin before they receive ethics clearance.

c) **Annual Renewal/Project Status Report** Ethics clearance is for only one year. The minimum requirement for renewing clearance is the completion of a "Annual Renewal/Project Status Report" in advance of the (1 year) anniversary of the original ethics clearance date. "

PLEASE NOTE:
It is the investigator's responsibility to complete the Annual Project Status Report that is sent each year by email 8 weeks in advance of the anniversary of the original ethics clearance to comply with the Research Integrity Policy. If ethics clearance expires the Research Ethics Board is obliged to notify Research Finance who in accordance with university and funding agency regulations will put a hold on funds.

27. **Additional Information:** Use this section or additional page(s) to complete any part of this form, or for any other information relevant to this project which you wish to provide to the Research Ethics Board.

28. **POSTING OF APPROVED PROTOCOLS ON THE RESEARCH ETHICS WEBSITE**
    a) It is the policy of MREB to post a list of cleared protocols on the Research Ethics website. Posted information usually includes: title, names of principal investigators, principal investigator department, type of project (i.e. Faculty; PhD; Masters, Undergraduate etc.)
    b) You may request that the title be deleted from the posted information.
    c) Do you request that the title be eliminated from the posted information? **[ ] Yes [x] No**
    d) The ethics board will honour your request if you answer **Yes** to the above question **27 c**) but we ask you to provide a reason for making this request for the information of the Board. You may also use the space for any other special requests.
    e) < List of MREB Cleared Protocols > < List of Undergraduate SREC Cleared Protocols >

N/A

Supporting Materials Checklist:

Instructions:

Complete this checklist to identify and describe your supporting materials to ensure your application form is complete

- *When supplying supporting materials, ensure that they are properly labeled (e.g., "Appendix C: Interview Guide for Teachers") and referenced in your protocol (e.g., "The interview guide for teachers – see Appendix C – is...").*
- *Do not cut and paste supporting materials directly into the application form; submit each as a separate appendix.*
- *If you have multiple supporting materials of the same type (e.g., multiple letters of information that target different populations), list each supporting material on a separate row in this checklist. Add a new row to the table if necessary.*

Supporting Materials Checklist	**I will use this type of material in my study** *(Insert X below)*	**I have attached a copy of this material in my protocol** *(Insert X below)*	**This is how I labeled and titled this material in my protocol** (e.g., *Appendix A – "Email Recruitment Script for Organizational Workers"*)

Supporting Materials Checklist	**I will use this type of material in my study** ***(Insert X below)***	**I have attached a copy of this material in my protocol** ***(Insert X below)***	**This is how I labeled and titled this material in my protocol** **(e.g., *Appendix A – "Email Recruitment Script for Organizational Workers"*)**
Recruitment Materials			
Study Information Brochure			
Video/audio recording that explains study details			
Participant Screening Form			
Recruitment Advertisements			
Recruitment Poster			
Recruitment Script – Verbal/Telephone			
Recruitment Script – Email (direct to participant)	x	x	Appendix A
Recruitment Script – Email (From holder of participant's contact information)			
Recruitment for follow-up interview			
Snowball Recruitment script			
Reminder/thank you/ card/script/email			
Appreciation Letter/certificate – For Participants			
Other			
Informed Consent Materials			
Consent Log (to record oral consent)			
Oral/Telephone Consent Script			
Letter of Information & Consent Form – **Participants**	x	x	Appendix C
Letter of Information & Consent Form – **Parent**			
Letter of Information & Consent Form - **Guardian** or **Substitute Decision Maker**			
Letter of Information & Assent Form – **Minors**			
Online survey brief information/consent and implied consent buttons			
Letter of Support for Study			
Research Agreement			
Other			
Data Collection Materials			
Information Sharing/Data Access/Transfer Agreement (for secondary use of data)			
Demographic form - Participant's			
Instructions for participants			
Interview Guide – (Questions for face to face, telephone, Internet/email interview)	x	x	Appendix B
Interview Guide – Questions for Focus Groups			
Questionnaire or Survey questions & instructions (Paper and pencil or online formats)			
Rating Scales/inventories/Assessment Instruments			
Role-play/simulation scripts			
Stimuli used to elicit responses			
Images (photos, diagrams etc.) depicting instruments, equipment, exercises etc.			
Other			
Debriefing Materials			
Debriefing Form			
Deception Study - Debriefing Letter & post debriefing consent form			
Deception Study- Debriefing script – verbal			
Other			
Confidentiality Materials			
Confidentiality Oath/ Agreement			
Confidential Study Code Key Log			
Other			
Materials for previous review by other REBs			
Application form –Other REBs (Original)			
Application form – Other REBs (Revised)			
Communication between REB & researcher			

Supporting Materials Checklist	**I will use this type of material in my study** ***(Insert X below)***	**I have attached a copy of this material in my protocol** ***(Insert X below)***	**This is how I labeled and titled this material in my protocol** **(e.g., *Appendix A – "Email Recruitment Script for Organizational Workers"*)**
(letters, emails, faxes etc.)			
Clearance Certificate (Other REBs)			
Other			
Other Supporting Materials			
Compensation Log			
List of support services for participants			
Participant Appreciation - letter, script, email or certificate etc.			
Researcher Training Certificates			
Scientific Licenses			
Other			

29. Researcher Assurance: < SECTION G – SIGNATURES >

[x] I confirm that I have read the McMaster University Research Integrity Policy http://www.mcmaster.ca/policy/faculty/Research/Research%20Integrity%20Policy.pdf , and I agree to comply with this and other university policies, guidelines and the Tri-Council Policy Statement (TCPS) and of my profession or discipline regarding the ethical conduct of research involving humans.

[x] *In addition*, I understand that the following *all constitute violations of the McMaster University's Research Integrity Policy*:
- failure to obtain research ethics clearance;
- carrying out research in a manner that was not cleared by one of the university's REBs;
- failure to submit a **Change Request** to obtain ethics clearance prior to implementing changes to a cleared study;
- failure to report an **Adverse Event** (i.e., an unanticipated negative consequence or result affecting participants) by the investigator or faculty supervisor of student research to the MREB secretariat and the MREB chair, as soon as possible and in any event, no more than 3 days after the event occurs;
- failure to submit an **Annual Renewal/Project Status Report** in advance of the 1 year anniversary of the original ethics clearance date.

Signature of Faculty, Student or Staff Researcher **PLEASE PRINT NAME HERE** **Date**

Signature of Faculty, Student or Staff Researcher **PLEASE PRINT NAME HERE** **Date**

Signature of Faculty, Student or Staff Researcher **PLEASE PRINT NAME HERE** **Date**

Supervisor Assurance for Graduate or Undergraduate Student Research:

[x] "I am the supervisor for this proposed student research and have read this ethics application and supporting documents and deem the project to be valid and worthwhile, and I will provide the necessary supervision of the student(s) researcher(s) throughout the project including ensuring that the project will be conducted as cleared and to make myself available should problems arise during the course of the research.

Signature of Faculty Supervisor of Student Research **PLEASE PRINT NAME HERE** **Date**
(Add lines for additional supervisors.)

The signature page may also be emailed as a scanned PDF or be sent by campus mail to GH-305.

Appendix A

Email Recruitment Script
Spencer Smith,
Associate Professor, Computing and Software
Impact of Document Driven Design in Scientific Computing

Subject line: McMaster Study – Impact of Document Driven Design in Scientific Computing

Would you be interested in providing me with a short computational program that you have written, or that you use in your work, for re-implementation by a student team? We are currently looking at using a series of case studies to research the effectiveness and practicality of a document driven development process for scientific software. In addition to providing your code, you will be asked to participate in two interviews (0.5 – 1.0 hour each) and several (3 – 6) short sessions (about 0.5 hour each) to answer technical questions and to discuss the new documentation. In no event will your total time commitment over the course of the study exceed 5 hours.

We selected your name because your research work, as shown on your web-page, likely includes the development of scientific computing software. Participation in our interviews is unlikely to cause harm or discomfort. However, there is a chance that you will feel uncomfortable if you perceive that your current approach to software development is being criticized. Please be assured that this is not our intention. Our goal is to systematically measure the impact of a document driven approach, which may turn out to be inferior to your current approach. You can discontinue participation in this study any time. Full details on the study can be found in the attached letter of information.

This study has been reviewed and cleared by the McMaster Research Ethics Board. If you have any concerns or questions about your rights as a participant or about the way the study is being conducted you can contact:

The McMaster Research Ethics Board Secretariat
Telephone: (905) 525-9140 ext. 23142
c/o Research Office for Administration, Development and Support (ROADS)
E-mail: ethicsoffice@mcmaster.ca

We would like to thank you in advance for your time and consideration. After a week, we will send you a one-time follow-up reminder. Please do not feel any pressure to participate in this study. If you do not have time to be involved, we have many other potential participants that we can work with.

____

Spencer Smith, PhD, PEng
Associate Professor
Department of Computing and Software
McMaster University

Tel: (905) 525-9140 X27929
Room: ITB/167
Web: http://www.cas.mcmaster.ca/~smiths

DATE: May 26, 2015

APPENDIX C
LETTER OF INFORMATION / CONSENT

A Study About Document Driven Design for Scientific Computing Software

Principal Investigator:
Dr. Spencer Smith
Department of Comp. & Soft.
McMaster University
Hamilton, Ontario, Canada
(905) 525-9140 ext. 27929
E-mail: smiths@mcmaster.ca

Student Investigator:
Thulasi Jegatheesan
Department of Comp & Soft.
McMaster University
Hamilton, Ontario, Canada
NA
E-mail: jegatht@mcmaster.ca

***Co-Investigator(s)**:*
Dr. Diane Kelly
Associate Professor
Department of Mathematics and Computer Science
Royal Military College of Canada

Purpose of the Study: We are interested in your opinions on the application of software engineering methods, principles and tools toward improving the quality of scientific computing software. In particular, we are trying to determine whether a document driven approach improves the reliability, maintainability, reusability and verifiability of scientific software.

Procedures involved in the Research: For the study you will be asked to contribute a scientific program of interest to you. This program will be redeveloped by my team, using a document driven process. To measure the effectiveness of the document driven process, you will be interviewed twice by Dr. Diane Kelly: once at the beginning of the project and once at the end. Each interview is expected to last between 30 minutes and one hour. The interviews will take place on the telephone or via Skype. Your responses will be audio recorded by Dr. Kelly and later transcribed by her. The purpose of the first interview will be to determine background on you and your software, as well as your current attitudes toward software development and documentation. A sample question would be, "What do you do to build confidence in your code?" The purpose of the second interview is to assess the impact of the newly developed software. A potential question would be, "Will this experience influence how you develop software?" The full list of questions is attached to this letter.

In addition to the time for the interviews, we will require some of your time to discuss your software. The discussion will involve answering technical questions related to your software and going over the new documentation. We anticipate needing about half an hour per week for 3 to 6 weeks. Depending on your preference the questions can be answered in-person, over e-mail, over Skype or over the telephone. Your maximum time commitment over the course of the entire project will be capped at 5 hours (two 1 hour interviews and six 0.5 hour question answering sessions.)

With respect to the ownership of the original code and the redeveloped code, the licence will stay the same as for the original code. That is, if the original code is under an open source license, this will continue to be the case and the redeveloped code will automatically come under the same open source license. If the original code is proprietary, this same status will be maintained in the new code. If you require a Non Disclosure Agreement (NDA), then this agreement will be respected by the researchers and student developers. You are considered to be the owner of the original and the redeveloped code. The researchers will not share the code, or use it for any purposes other than for this project, unless the original license permits such use.

Potential Harms, Risks or Discomforts: Participation in our interviews is unlikely to cause harm or discomfort. However, there is a chance that you will feel uncomfortable if you perceive that your current approach to software development is being negatively criticized. Please be assured that this is not our intention. Our goal is to systematically measure the impact of a document driven approach, which may turn out to be inferior to your current approach. If there is any question you do not feel comfortable answering, you do not need to answer.

There is also the social risk that your reputation might be damaged if someone deduces your identity and concludes that your code Is not well written. Given our efforts to keep you anonymous, it is unlikely that you will be identified by anyone. In the unlikely event that you are identified, there will be no basis for criticism, since the new documentation that is being created is not currently used by other scientific software developers. The approach used to redevelop the code may turn out to be a recommended approach in the future, but this is not currently the case.

Potential Benefits: The research will potentially benefit you through the redevelopment of your software. At the end of the project you will have a new version of your software that is fully documented and tested. You maintain ownership of your original code, and of the redeveloped code. You may choose to use the new software, use portions of the new software, or stick with your original version.

Confidentiality: Every effort will be made to protect (guarantee) your confidentiality and privacy. We will not use your name or any information that would allow you to be identified. Only the research team and myself will know your identity. In any publications generated from this work, we will use generic terms to refer to you and your software program. However, there is a chance that others might deduce your identity by combining the various generic facts.

The notes and audio recording of your interview responses will be kept in a password protected files on a password protected computer. We will retain the data for two years from the conclusion of the study so that it is available for preparing publications. After this time period the data will be destroyed. The original and redeveloped code will also be destroyed at this time, unless the original license permits retaining them.

Participation and Withdrawal: Your participation in this study is voluntary. If you decide to be part of the study, you can stop (withdraw), for whatever reason, even after signing the consent form, or part-way through the study. If you decide to withdraw, there will be no consequences to you. In cases of withdrawal, any data you have provided will be destroyed, unless you indicate otherwise. If you wish we will complete the software redevelopment, even if you withdraw from the study. If you do not want to answer some of the questions, we can skip those questions, and still keep you in the study. As part of the process for ongoing consent, you will verbally be asked to confirm your consent at each point of contact (interview of discussion session). Your verbal consent will be recorded in the table at the end of this document.

Information about the Study Results: We expect this study to be complete by January 2016. The results will be part of the MSc thesis produced by Thulasi Jegatheesan. If you would like a copy of the thesis, please let me know how you would like it sent to you.

Questions about the Study: If you have questions or need more information about the study itself, please contact me at: smiths@mcmaster.ca

This study has been reviewed by the McMaster University Research Ethics Board and received ethics clearance. If you have concerns or questions about your rights as a participant or about the way the study is conducted, please contact:

McMaster Research Ethics Secretariat
Telephone: (905) 525-9140 ext. 23142
C/o Research Office for Administrative Development and Support
E-mail: ethicsoffice@mcmaster.ca

CONSENT

- I have read the information presented in the information letter about a study being conducted by Spencer Smith of McMaster University.
- I have had the opportunity to ask questions about my involvement in this study and to receive additional details I requested.
- I understand that if I agree to participate in this study, I may withdraw from the study at any time.
- I have been given a copy of this form.
- I agree to participate in the study.

Signature: ______________________________ Date: ____________________

Name of Participant (Printed) ____________________________

Ongoing Consent at Each Point of Contact

Date	**Purpose of Contact (Interview or Discussion)**	**Consent Received (yes/no)**

Appendix B

Transcriptions of First Interviews

Interview Notes with Marilyn Lightstone - 21 October 2014

From Diane kelly

- <what code development do your students do?>
- most students run commercial codes
- some develop small simple models, about 1000 loc
-< guarantee quality of work?>
- verification - make sure what they believed they put in actually went in (as far as modelling data goes)
- check boundary conditions, ensure way using data is correct, build up problems in simple steps, compare to analytical solutions
- steep are: can you run the code correctly, can you develop the input correctly, then run the code and analyze results
- do a simpler version of study, e.g. laminar flow - plot curve of results from computer and compare to analytic results - should be identical
- <what challenges do students find when using codes?>
- creation of grids, learning the commercial tools (use tutorials from commercial company)
- best source of information: other people in lab
-infrequent use of commercial tool is challenging since eth user interface changes every couple of years
- commercial tool is MODTURC; expensive
- online user group exists but not useful
-students work on projects built on previous student's projects; hire graduate for a few months to pass their knowledge on to next student
-students generally leave behind Readme files but not much else
- have all their input files
- programs not difficult to run
- <reproducibility?> studies are reproducible forever in MODTURC - underlying principles don't change - just the user interface
- codes are in FORTRAN
- most students leave behind their final results, code (sometimes commented), no user docs
- Marilyn provides examples with comments to encourage commenting
- <trust codes such as MODTURC?>
- basically, no - found bugs
- users must do validation always - insist students always validate
- compare to published results, find test cases that exercise new parts of modules
-<do you use open source/freeware/otehr people's code?>
- have in past because MODTURC couldn't do something we needed
- had to validate it first
-huge difficulties because documentation not good, weird bugs, too complicated, needed to be really good at coding
- happy to share code with others
-<in journal publications, what do reviewers look for?>
- a results is sensible - looks for physical properties eg. adiabatic boundary condition contours hitting wall at 90 deg

- don't need to provide code
- include validation performed in write-up of paper
- <your reaction to being handed a new set of tools to use?>
- not that thrilled
- students may be interested
-have limited time, so the tool has to provide a good benefit and good payoff
- cannot be cumbersome
-<ideal code and documentation?>
- well commented; clear on how to use; explain input and output; confident code is correct; want to understand code; understand underlying assumptions
- <what would you consider a requirements document?>
- most students produce a theory "manual" for their work, describe discretization, etc.
-<would you include this with the code/part of the code?>
- no, better as a separate document, could have a link between theory and code such as variables used for each

Diane's impressions:
- codes written by students are very small
- everyone very focused on physics, not the code
- could be ok if the codes are simple
- have difficulty getting the students to write code comments - they write the theory up ok since that's what their theses are about
- a standardized template for documentation may be useful (though it still may be difficult to get a student to write decent comments)
- tools would be welcomed only if clear what their advantage would be - can't be cumbersome or time consuming, especially if it takes away from doing the science
- requirements considered the underlying theory and the problem statement - the underlying theory may be developed as they go
- don't seem to read the code very much - often code not available - depend heavily on various types of testing

Interview with [redacted] - Monday 17 August 2015

Code: [redacted]
- [redacted]
- written in 2007
- about 5000 loc
- current public version is FORTRAN 77; plans to release a FORTRAN 90 version in a month or two
- maintained by [redacted] and colleague in [redacted]
- can be run on a single processor, depending on size of data
- sample run is 500 galaxies, 40 cores on HPCVL and takes a week
- matrices being diagonalized can be anywhere from 20 rows to 2000 rows
- users are all over the world, eg. South Africa, Australia, etc.
- about 10 users at any one time - very specialized software
- released as open source where all components are fully available without commercial licences needed; problems with commercial licences when going to other countries
- maintain LINUX and DARWIN (MAC) versions of executables - most users use these instead of creating own executables, but full source code is available

Background of developers:
[redacted]:
- theoretical astro physicist who runs a lot of computational codes
- no formal software background but codes a lot and very cleanly
[redacted]:
- acts mainly as user/tester/designer/user support/bug identifier
- sits on observer side of software
- self taught programmer
- PhD was programming in FORTRAN 77
- also writes code using IDL to manipulate large astro images
- writes scripts to plug in libraries such as Numerical Recipes

Documentation
- user documentation mostly by [redacted]
- started by writing a paper about the software, then decided to make the software available publically, then upgraded the documentation to assist the users
- assumes user has necessary physics background
- describes input and output file formats, options; bulk of documentation are examples of using the software
- describes science for code
- use math symbology developed in one of the published papers
- [redacted] prof puts in code comments and uses good parameter names; keeps code very clean
- only two profs touch code
- include a version history in code listing changes included in the current version
- no design or requirements documentation

Version control

- use version control on public releases
- limited distribution of in-between releases for specific users for very specific problems
- keep all files for these in-between releases, using a manual systems based on dates
- official releases once every 2-3 years

Bug tracking
- have a small number of users, so a bug database not necessary
- number of bugs is 0-5 a year; may hear from someone with a bug every 6-8 months
- [redacted] is primary contact
- usual procedure is to address bug as soon as it is reported; use text file to record details of bug; track down possible cause and fix; report to [redacted] prof; he provides formal fix and [redacted] tests and approves it for distribution

Use of other software
- use linpac and numerical recipes codes, but can't distribute these as part of open source package
- use CFITSIO to read special input file format (FITS) from astronomy sources
- ran the code through a FORTRAN compliance checker in 2010; wasn't useful; mostly write code that is standard compliant
- have to be careful that whatever is used is freely available in other countries, eg. FORTRAN compilers

Quality concerns
- scientific integrity
- changes to code motivated by scientific return
- make code easier to use and make it correct
- improvements motivated by new scientific questions
- not funded to do software engineering
- backwards compatibility is important ; keep adding new stuff but ensure input text file not changed; users don't want to be worried about new releases every week - rather have infrequent releases timed like extent of a Master's student's length of study; 2 year old input file should still work

Testing
- test to ensure new code fixes problem - both [redacted] involved; [redacted] ensures code changes work for user
- [redacted] does regression testing by running through all documented examples to ensure results are reasonable
- couldn't possibly test all combinations of options available in code
- when hear from uses and students about problems, do further testing to check the problems out
- users themselves run simulations to check parts of code
-hard to have something insidious in this code that won't show up in tests - nature of computations in code
-not had a problem with reproducing a bug
- work closely with user and address problems immediately
- do limited code reading, more to check changes to a section of code that is familiar (ownership?)

[redacted] - Friday 21 August 2015 - Interview #1

Developer software background:
- worked in MATLAB in undergrad
- a few months ago developing software ([redacted]) for a company
- before this, would develop scripts to expedite process for writing research papers
- new software is in FORTRAN and Visual Basic (VB)
- learning as they go
- this is first time writing software to distribute to someone else
- first time to have to take care of issues didn't have to before - such as concern about ignorance of user, eg. all possible things that the user can make go wrong

Software: [redacted]
- 1300 loc of FORTRAN
- 1000 loc VB
- FORTRAN used to science and math
- EXCEL spreadsheet for user interface
- VB to transfer data from spreadsheet to FORTRAN program
- [redacted] is the only developer
- software calculates the resistance of a glass window subjected to the pressure of an explosion
- help user to design a pane of glass
- input is size of window, thickness of glass, type of glass, and explosive charge

Documentation:
- might have to write small user's manual for client - not discussed yet
- a presentation to users should suffice
- users only need to know spreadsheet and the ASDM Standard - don't need to know mathematics or science (ie. FORTRAN part)
- uses code comments - blocks of code commented, titles in subroutines - not thoroughly commented
- variable names are set up as a hierarchical structure - eg. input data - geometry - side A,B,C
- finding names of variables are getting too long and clumsy, so using pseudonyms for the longer names and it's not working well
- other documentation types not familiar with - eg. requirements docs, designs docs
- [redacted] will maintain - not sure of lifetime of code; will be obsolete if the ASDM Standards change - may be around for a few years - only need to document enough for [redacted]

Tools used:
- compiler
- issue tracking irrelevant because code is not old enough
- version control done by hand - once have a stable version, number it next in sequence and use this numbering system to identify all parts of the version - have a file directory set up to handle this
- have had no problems picking up earlier versions of software

- future work - will use random number generators
- used some stuff from the web for VB - eg. write a counter to check performance of subroutine
- don't use anything that is critical
- if used free software for something critical, would need to check it carefully
- recently had trouble with VB and EXCEL since had to upgrade to newest release - broke some of existing VB code - took several days to find the problem - backwards compatibility a problem

Testing:
- compare output with examples worked out by hand
- use software to generate design charts in the Standard - if match, means software is working - realizes this isn't complete checking, but addresses the core (FORTRAN?) part of the code that it is working properly
- company will do additional hand calculations to check software and will report back any problems
- regression testing? <explained what this is>   - will need this in the future - too soon now
- when have time, check out routines individually - work out hand examples and check the code matches, then incorporate into the rest of the software
- coming up with a system of error messages that report any problems with the calculations and subroutines misbehaving

[redacted] - [redacted] - Fri. August 21 2015 - [redacted]

Developer background:
- 1st year engineering undergrad used C# for simple GUIs and simple math/engineering programs
- used MATLAB in a numerical methods course
- learned Python for a course where they were teaching assistant
- used MATLAB in undergrad research terms
- now use MATLAB on a daily basis

Software: [redacted]
- written in MATLAB
- 2 or 3 main components
- user interface is about 100 loc
- genetic algorithm for optimization is about 1300 loc including 150 lines comments, 20% white space = ~800 actual lines of code
- within optimization, there are routines to evaluate safety of candidate surfaces - about 500 loc
- started in summer 2010 as adaptation from an earlier code
- does a slope stability analysis: first code only used circular shapes; new code can work with non-circular shapes - substantial changes to old code so new code is essentially a complete rewrite
- usually [redacted] is the only developer - Spencer's student is only other developer
- code is strictly an instructional tool used in a foundational course
- can have about 30 undergrad students using it at one time
- code is set up to detect some non-sensical input, eg. obvious physical problems such as overlapping surfaces
- users expected to understand the underlying problem and not input nonsense data
- have not had students report any bugs or problems with software

Trust in software?
- uses established algorithms
- tested against textbook examples
- not intended as commercial package
- students are running it all the time, but never had a problem with the software - problems are students inputting bad data
- no random crashes, stable

Documentation
- meaningful variable names
- development notes
- comprehensive commentary in code
- uses MATLAB format of comments: uses MATLAB help template, follows that format
- students are given input/output variables for interface of program
- created tutorial document for students that includes worked example and high-level explanation of physics/math that's in code
- did one conference presentation on code

Tools?
- compiler - MATLAB environment
- GIT version control
- not using a unit testing framework: developed own test scripts and checked output against literature
- does regression testing by rerunning own test scripts
- no freeware, open source, etc. associated with this code
- hasn't had to recreate an earlier version, but could do it by picking up code from repository
- hasn't had to worry about recreating earlier results - code has not changed in such a way that results would have changed substantially

[redacted] Wed. 26 August 2015

Developer
- undergrad in [redacted] engineering at [redacted]
- did several programming courses; later ones in MATLAB
- worked at MDA Space Missions doing robotics
- PhD in neuroscience - models of hearing systems in brain - [redacted]
- used MATLAB
- [redacted]
- develops software to treat tinnitis using customized sound therapy - embed sound therapy in music

Software
- written in MATLAB
- incorporates computational model of hearing brain
- uses standard MATLAB functions
- written software is about 2000 loc
- started in September 2012
- minor changes for about 2 years
- has added some automatic pulling of data from input files
- inputs person's audio profile - possibly what changes have occurred to cause tinnitis
- uses theory of what might have happened
- uses different algorithms to alter music to try and reduce person's tinnitis
- developer works alone and is only user
- users are removed from software - use product of software but not software itself
- people submit audio profile online then pick up music files off server
- software takes about an hour to run to create a therapy

Tools used
- MATLAB environment
- runs on UNIX
- was using svn version control initially
- after started business, has time constraints and puts time efforts into running code and getting work done rather than housekeeping => not using svn right now
- issue tracking - issues show up only once in a while, so issue tracking tool not needed
- uses freeware to store sound theory as a wave file - UNIX based - then another freeware to convert this to MP3
- another tool is MP3 Gain to even out the volume of different tracks of music
- would eventually like a proper testing framework and version control in place

Documentation
- no user documentation - is own user
- code is commented to explain functions and procedures
- no outside publications because algorithm is a trade secret/proprietary
- in future, will need documentation if company grows and new developers hired on
- better commenting and external documentation would be desirable as well as a more formal

process

Increase trust?
- early on, did traces while running program, ie., used debugger in MATLAB and watched what code was doing under specific test cases
-everyone's output is different, so cannot predict what the program will give for a particular case => testing hard
- did code reading when developing - also went back and read code and updated comments when Spencer proposed this project
- regression testing not done formally - has gone back to check previous results to compare new algorithm or changes to an algorithm
- have metrics in model to check sufficiency of changes - allows comparison to previous results
- has done changes to improve how long the software runs to produce a therapy - has improved this significantly
- neural networks have random seeding which makes perfect reproducibility impossible - [redacted]
- [redacted]

[redacted] - [redacted] - 25 Aug 2015

Software: [redacted]
- tool to analyze large amounts of acoustic data; does visualization and automation of analysis
- 4 people working on the code
- 2 of these people are its current users
- 1 other individual using the code
- code is 2-1/2 years old
- didn't have a name until Spencer suggested a name :)
- 2 of current four people wrote the original code
- code is about 200 loc
- written in Python 2.76
- part of output from code goes to a commercial stats package: Procensus
- a lot of modifications going on from beginning
- continually evolving product and learning how to use acoustics in different ways
- some analysis is getting complex; moved from 2 dimensional to 4 dimensional space models; manipulating large amounts of data; visualization is important in analysis

Background of developers:
- all have degrees in chemical engineering
- 1undergrad course in programming in first year
- all developers are working on acoustics
- can program in MATLAB as well as Python

Documentation:
- when first created, wrote user documentation
- minor inline comments on blocks of code; eg. group of code for visualization
- code is part of a project for a company
- code is included in an appendix in documentation back to the company
- no other papers/reports/presentations on code

Building Trust in Code
- normal data is a mish-mash of frequencies
- to test/validate, create a single pulse frequency and check the code output within a given error bound
- had a lot of problems with code over time
- created a mirror code in MATLAB using some of MATLAB's canned library routines
- do direct comparisons with the MATLAB code
- can't use MATLAB for the product for the company: MATLAB too slow and costly because of it being proprietary
- asked about regression testing - seems code is too new and too small to take this into consideration
- any new routines are independent, eg new analysis, new visualization, and don't impact existing code base functions
- new routines are checked individually
- tied and true models/math are used - eg. Fast Fourier Transforms to get visualization

Tools
- Python interpreter plus downloaded packages such as a GUI interface for Python editing
- have hit problems with moving to Python 3.0 - their code is interfacing with a subroutine from National Instruments that does data logging, and Python 3.0 version won't work with the instrument code
- there is a large enough community using Python 2.76 that they have demanded continuing support for this version
- don't use any other open source of freeware
- version control done by hand using numbering system and file system; have a number for each version and separate file; increment number each time
- no formal problem tracking - small group that interacts closely - know what the answer should be and keep working on the code until the expected answer comes out
- no unit testing framework used
- reproducibility of results not a problem

Appendix C

Transcriptions of Second Interviews

Follow-up comments from [redacted]; 10 March 2016

[redacted] has read through the documentation but has not gone through the code nor has tested the new version. Impressions are from looking at the documentation and talking to Spencer and Thulasi.

Initial impression:

- most helpful was Module Guide (which [redacted] received last)
- reasons: much higher overview of software structure; good for non-expert perspective
- SRS is ground-up detail once a person "gets into" the code
- MIS gives technical detail

Questions (long form of questions read to [redacted] - short version reproduced here for reference):
1) What are the most noticeable differences?
2) What drove the greatest benefit from the redevelopment?
3) Will you use any of the redevelopment tools introduced?
4) Will your approach to requirements and design change?
5) Will your approach to V&V change?
6) Will this change how you develop software? Other benefits of this exercise?
7) Advantages and disadvantages of DDD process?

[redacted] Comments:

1) - found emphasis on "hiding" interesting : what user doesn't need to know: minimalist approach
-previously assumed user was expert and had some level of familiarity with code
-this new approach is better for a general user who wants to use the code in a sophisticated way but it new to the code
- doesn't see this new way a disadvantage to seasoned user - advantage across the board
- code is a multi-parameter non-linear fit and user needs to know enough to get started; if thrown in at "deep end", user can get misdirected very quickly
- need to ease the user in better; hide things they don't need to know right away

2) - benefits came from a combination of both the approach and extra eyes on the code
- this was first set of eyes on the code from a non-expert in the application domain
- outside perspective/fresh set of eyes on the code is helpful
- funding is for science not for software development
- code is more user friendly form basic perspective - helpful
- feels this is a powerful approach

3) - (considering the documents themselves as tools rather than application tools) keeping list of dependencies is useful
- trace between requirements and modules useful
- like to continue this going forward
- have done version control in the past

- only do new version releases every 1 to 2 years, so users aren't faced with new versions every time they sign on
- relatively small bugs - no new versions - batch changes

4) - try to take some elements of what was done forward
- module guide kept up to date is important
- SRS and MIS are more complicated and more difficult to keep up to date
- SRS and MIS probably won't change until some fundamental changes happen to the code - then these changes happen more organically; hard to justify students (physics) making these changes to the documents without the support from the science side of things

5) - didn't talk about testing
- testing done right now includes regression testing with standard simulation cases
- haven't regression tested the new version yet

6) - interesting to see someone with a different viewpoint of the code
- notion of hidden modules; hiding information from users where they don't need it; keep this in mid
- not going to code in a fundamentally different way from way currently used
- keeping track of which modules interact with each other and with the requirements will continue to be useful

7) - don't know how useful DDD will be
- scientists do science eg., make software, see if it works, use it, distribute it, work backwards from it
- DDD gives you better code but requires foresight on what you're going to use the code for; this doesn't work in reality; to use DDD almost need to start all over again and we aren't funded to do this
- fundamentally it's a time thing and a money thing; the Canadian funding structure doesn't support a scientist doing software

Follow-up Comments from [redacted] - 7 April 2016

Initial discussion:

- looked at SRS and Module Guide
- looked at code and ran a couple of routines to check output
- not familiar with Python

Question 1:

- output format changed
- Excel interface replaced with Python interface
- Excel interface worked ok but tends to break down with new versions; has to be tweaked
- wants to build new interface, explore something different than what came from redevelopment
- structure/ architecture slightly different
- is better organized than before
- if go through each subroutine, have problem understanding - there is more commenting but not as explanatory as could be (possibly because those doing redevelopment didn't understand scientific background sufficiently)

Question 2:

- benefit is from more systematic approach to architecture
- had no systematic approach before
- just wrote code
- architecture has changed from first version (a year ago) to now

Question 3:

- support tools
- talked briefly with Spencer about support tools, eg. using GIT for versioning
- already aware of versioning tools
- currently doing small projects alone - "don't need a canon to shoot a fly"

Question 4:

- Module Guide
- when looked at map of hierarchy of modules, arrows seemed to be backwards
- eg. control module points at input module
- assume this is top down type of diagram
- usual thinking is bottom up
- (suggested that the difference may be more control flow versus data flow - I think that is how [redacted] usually thinks about their code)
- [redacted] suggested map of modules would be more useful restructured as Venn diagrams

- SRS

- most of SRS is explanations of things "I know well" (quote from [redacted])
- provided systematic way of organizing information
- data constraints table
- useful idea for a table
- some of the easy constraints are there
- some constraints depend on results of specific calculations and these are not mentioned in the table
- (I asked about keeping the SRS and MG up-to-date)
- [redacted] didn't look into how to generate these two manuals
- would like to keep them up-to-date
- wouldn't do it if it takes too much time

Question 5:

- I asked if testing was discussed with Spencer - they couldn't remember
- I asked about regression testing - they do this already, runs a handful of tests (4), but not done systematically
- I asked if approach to testing has changed: answered, no; hasn't learned a systematic way to test

Question 6:

- I asked how they would change his approach to software development
- small project, wouldn't bother changing
- bigger project (eg. 10k lines or so) would start with top-down approach and think about architecture up front

Question 7:

-I asked about advantages and disadvantages of applying DD process
- disadvantages
- adding a layer of abstraction that he could do without
- there is a measure of jargon that is a barrier
- handling documents is too time consuming for a small project
- was hoping to get a better way to structure his interpolation algorithm - right now it "looks like a lump"; after restructuring, still "looks like a lump"
- advantages
- learned some Python
- future maintainability of projects should be better
- at end of module descriptions has examples of all data handled by module, ie. includes possible numbers in an array

Suggestion:
- right now get web pages with explanations of modules and call structures
- more useful would be an interactive map of whole system where you could click on specific modules and get more details, then click on details (eg. variables) and get even more details

[redacted] - Tuesday 22 March 2016

Background:
- from Spencer and students, received:
- reorganized code
- SRS
- Module Guide
- Interface Specification
- testing documentation
- activities so far
- has downloaded software and looked at it
- not had a chance to work with the code yet
- has run set of tests that Spencer and students developed

1) most noticeable difference
- software divided into significantly more modules
- had 6 modules previously and 5 or 6 test scripts
- now have 10 to 12 modules
- split out aspects of what code does into separate modules
- each modules does one specific thing
- larger number of test scripts
- now have automated testing process for wider number of types of tests
- tests now cover what was previously considered "trivial" things that were tested "on the fly" and not recorded, eg. input format now tested

2) greatest benefit
- program itself same as before (from black box perspective) since no new functionality added
- documentation provides greater amount of detail
- good for sharing information on design choices which are in your head - get this down on paper
- even for author himself, helps to remember design choices
- in future, if have students changing code, documentation will be useful in bringing others up to speed
- from aspect of someone spending time on project: DDD approach helped to guide them and make their output more useful

3) software support tools introduced
- already using source control tool
- testing docs used LATEX as part of automated doc generator
- may use this in future on a project that is sufficiently large, eg. one to several year project

4) going forward with new documentation
- would continue to use on larger projects that are shared with other developers
- can see usefulness but it's a matter of convincing others it's worth the time
- will try to keep documents up-to-date as project ([redacted]) goes forward
- adds extra work
- forces you to specify what program does ahead and hold yourself accountable to this

5) going forward with testing
- already doing regression testing
- already doing V&V in a similar way
- now have more detailed unit tests
- exercising internal operation of program
- see value in automating this
- will add this approach to doing unit testing of software
- learned to focus on unit testing

6) going forward with software development approach
- will have this type of development in mind
- will depend on scale of project
- if project is larger time scale, large group of users and/or large group of developers
- having some sort of documentation is indispensable
- framework presented here makes sense
- tool support
- repository system and auto testing useful for larger projects
- benefits?
- have now seen a more organized approach to regression and unit testing
- easier to maintain and so more likely to maintain it

7) advantages and disadvantages of DDD
- advantages
- high level of detail recorded so don't lose knowledge capital that has been built up
- record what program is supposed to do and then check that it does it
- spending initial time up front is useful
- code is reorganized for "design for change"
- documentation is "design for change" at a more abstract level
- disadvantages
- up front extra time to development process seen as a hurdle
- for a small project, seen as unnecessary

[redacted] - tinnitus project - Interview Monday 14 March 2016

Questions for interviewer's (Diane) understanding of what was done for [redacted] project:

- SRS delivered
- written by Spencer and students
- based on discussions and meetings with code owner
- testing report delivered
- summarized test cases
- data and expected output came from code owner
- Spencer and students designed test scripts
- hazard analysis delivered
- related to project as a whole, not just software
- looked at data + output + hazards related to use of output products of software
- more focused on use of software
- ie ensuring correct input from hearing clinic, proper usage of end product
- did some exploratory work to measure risk but no objective measure was found

Note: Questions 1 and 2 were omitted because code was not redeveloped due to IP concerns

Question 3:
- version control was used to some extent previously; going forward, definitely use
- other tools?
- the way testing was set up is valuable and will be used in future for further testing

Question 4:
- requirements document: have integrated it into a more comprehensive FDA required report
- FDA has a similar standard for their approved documents
- found the exercise of creating the SRS useful as a training exercise for doing FDA documents
- the SRS will be maintained as part of the required FDA documents
- plan to do guidelines on how to keep the SRS/FDA document updated in order to aid future owners of this software system

Question 5:
- in future, will give more emphasis on traceability in testing
- will design testing to focus on hazard analysis
- previously not doing formal testing; will organize testing at a high level
- better understanding now about when and the type of testing that should be done
- found testing exercise useful
- got different view of software
- better insight into limitations of software and what might have to be done in future

Question 6:
- better appreciation of formal methods and testing
- better appreciation of documentation that could be done in advance of coding
- look at idea of bringing someone into project to do software development instead of code

owner doing it and having to clean it up afterwards
- motivated by Spencer's project, the code owner did some changes to the code
- some changes were motivated by advice from Spencer
- some code changes were decided by code owner
- owner's goal was to remove redundancies and improve documentation on how the code works

Question 7:
- advantages of the DDD approach:
- forces you to put effort into design and planning to get cleaner design
- write the code to make testing easier later
- DDD fits in with FDA standards and got a better sense of that
    - FDA standards requires that you do requirements and test planning ahead
- would definitely follow DDD method
- disadvantages of the DDD approach
- as part of a start-up, can eat up resources and time and you need to find out what works quickly
- had to do some massaging to get the DDD document to fit the FDA standards
- had hoped to have a consultant on board to guide Spencer and students on what was required for the FDA standards, but was unable to make that happen
- need a team of people to use the DDD approach: challenging for one person to play all roles and get things done