PERCEPTIONS OF SE APPLIED TO SC

INVESTIGATING COMMON PERCEPTIONS OF SOFTWARE ENGINEERING METHODS APPLIED TO SCIENTIFIC COMPUTING SOFTWARE

By
MALAVIKA SRINIVASAN

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 (2018)
(Computer science)

McMaster University
Hamilton, Ontario

TITLE: Investigating Common Perceptions of Software Engineering Methods Applied to Scientific Computing Software
AUTHOR: Malavika Srinivasan
CO-SUPERVISORS: Dr. Spencer Smith, Dr. Sumanth Shankar
NUMBER OF PAGES: vii, 91

Abstract

Scientific Computing (SC) software has significant societal impact due to its application in safety related domains, such as nuclear, aerospace, military, and medicine. Unfortunately, recent research has shown that SC software does not always achieve the desired software qualities, like maintainability, reusability, and reproducibility. Software Engineering (SE) practices have been shown to improve software qualities, but SC developers, who are often the scientists themselves, often fail to adopt SE practices because of the time commitment.

To promote the application of SE in SC, we conducted a case study in which we developed new SC software. The software, we developed will be used in predicting the nature of solidification in a casting process to facilitate the reduction of expensive defects in parts. During the development process, we adopted SE practices and involved the scientists from the beginning. We interviewed the scientists before and after software development, to assess their attitude towards SE for SC.

The interviews revealed a positive response towards SE for SC. In the post development interview, scientists had a change in their attitudes towards SE for SC and were willing to adopt all the SE approaches that we followed. However, when it comes to producing software artifacts, they felt overburdened and wanted more tools to reduce the time commitment and to reduce complexity.

While contrasting our experience with the currently held perceptions of scientific software development, we had the following observations: a) **Observations that agree with the existing literature:** i) working on something that the scientists are interested in is not enough to promote SE practices, ii) maintainability is a secondary consideration for scientific partners, iii) scientists are hesitant to learn SE practices, iv) verification and validation are challenging in SC, v) scientists naturally follow agile methodologies, vi) common ground for communication has always been a problem, vii) an interdisciplinary team is essential, viii) scientists tend to choose programming language based on their familiarity, ix) scientists prefer to use plots to visualize, verify and understand their science, x) early identification of test cases is advantageous, xi) scientists have a positive attitude toward issue trackers, xii) SC software should be designed for change, xiii) faking a rational design process for documentation is advisable for SC, xiv) Scientists prefer informal, collegial knowledge transfer, to reading documentation, b) **Observations that disagree with the existing literature:** i) When unexpected results were obtained, our scientists chose to change the numerical algorithms, rather than question their scientific theories, ii) Documentation of up-front requirements is feasible for SC

We present the requirement specification and design documentation for our software as an evidence that with proper abstraction and application of "faked rational design process", it is possible to document up-front requirements and improve quality.

Acknowledgments

I shall begin with my spiritual master ‘Sai’; I am presenting this work at his lotus feet. Without his will I would have never accomplished anything in my life including this Master’s degree.

I would like to thank all the people who contributed in some way to the work described in this thesis. First and foremost, I would like to express my sincere thanks and gratitude to my co-supervisor Dr. Spencer Smith for his valuable guidance, motivation, patience, cooperation and continuous support throughout this program. His valuable guidance and feedback helped me immensely to improve my knowledge in the field of study and specifically in writing and documentation. I feel lucky to have him as my supervisor. It would never have been possible for me to take this work to completion without his incredible support and encouragement.

Next, I would like to express my sincere gratitude and thanks to my co-supervisor Dr. Sumanth Shankar for his valuable guidance, patience, financial support and cooperation with the scientific aspects of this project. I benefited greatly from many fruitful discussions we had in regards to this thesis. I must not forget to thank him for believing in me and providing this opportunity to pursue this graduate studies.

I must express my gratitude to Dr. Balamurali Kannan, my husband, for his continued and unfailing love, support and understanding during my pursuit of this master’s degree. I am always thankful to him for giving me the liberty to choose what I desired and pursue my dreams. This master’s would not have been possible without his support and respect for my life choices.

I must express my gratitude to my mother, Chitra for her support, patience and hard work in taking care of my son in my absence. I am truly blessed to be her daughter.

I dedicate this thesis to my son, Ramanathan, who is my pride and joy of life. I love him more than anything and I appreciate all his patience and support during mommy’s master’s studies.

I am also thankful to my father, Srinivasan for being my role model in perseverance and dedication. His support and advices has provided me the strength to overcome my difficult times. My special words of thanks should go to my brother, Hariharan for his constant encouragement and invaluable support and humor over the years.

Last but not the least, I would like to thank my grandparents and my maternal uncle for their help and support throughout my life. These people have played a major role in encouraging me to pursue my dreams and have been a pillar of strength for my education. This accomplishment would not have been possible without their blessings.

Contents

Chapter 1

Introduction

Scientific Computing (SC) is the collection of tools, techniques, and theories required to computationally solve mathematical problems in science and engineering [22]. Examples of typical scientific computing software are CMAQ (Community Multi-scale Air Quality Model) [3], and SEDA (Statistical Earthquake Data Analysis) [47]. SC software has gained importance in the last three decades, moving science "from test tubes into silicon-based simulation" [91]. According to Ahmed and Zeeshan [4], SC software is used for "processing, analyzing, visualizing, managing, sharing, experimenting and in some cases even generating new raw data". SC provides scope for research in otherwise impossible conditions [71], such as simulation of a nuclear explosion. SC software has a crucial role in scientific research.

The results from SC software are often used in fields such as nuclear engineering, medicine, climate predictions and the military [15, 38, 66]. For example, in simulations of nuclear weapons, code is used to determine the impact of modifications, since these weapons cannot be field tested [61]. At the same time, even small software, faults such as one-off errors, have caused the loss of precision in seismic data processing programs [23]. Considering the societal impact of SC applications, it is critical to ensure that the SC software is faultless and of the highest quality. According to [19], "there is a growing concern about the reliability of scientific results based on ... software". Several highly influential articles had to be retracted and more than five years of work was lost as a result of a trivial programming error in a previous researcher's work [52]. NIST (National Institute of Standards and Technology) claims that software bugs, or errors, are highly prevalent and detrimental, that they cost the U.S. economy an estimated $59.5 billion annually, or about 0.6 percent of the gross domestic product" [54]. "The time and effort needed to fix errors increases exponentially the later they are identified" says Alden and Read [5]. Hence, it is critical to pay attention to the quality of applications by detecting and eliminating errors as soon as they are identified.

As mentioned earlier, there is a growing concern about the quality of SC software. A few areas of concern include testability, reproducibility and reusability. The need to improve testability has been highlighted by Kanewala and Bieman [36] in their recent article, which says “Scientific software presents special challenges for testing due to the characteristics of the scientific software and the cultural differences between scientist developers and software engineers”. The need for improving reproducibility for SC software is increasingly recognized, with various conferences, workshops and individuals calling for change [6]. Mozilla Science Lab is running an experiment by reviewing selected pieces of code from published papers in computational biology, with the aim to improve the quality of researcher-built software. According to Kaitlin Thaney, the Lab director of Mozilla [25], “Scientific code does not have that comprehensive, off-the-shelf nature that we want to be associated with the way science is published and presented”. The problems with reproducibility are also highlighted by a recent study of the code for 402 computer systems papers in which only 48.3% of the code was both available and compilable [12]. It is evident that, there is room for improvement in the current approach to developing SC software.

The primary goal of our research is to improve the quality of SC software. According to Roy [64], the alarming results about the reliability of scientific software in a case study conducted by Hatton [23, 24] highlight the need for employing good SE practices in SC. Developers of scientific software usually perform validation to ensure that the scientific model is precisely applied to the physical phenomena of interest [37], using primarily mathematical analyses [61]. But they rarely perform systematic testing to identify faults in the code [30, 38, 41]. These points create a strong demand to have a process to develop scientific software, where the process improves the quality aspects like verifiability, reproducibility, reusability, and maintainability.

In a previous attempt by Smith et al. [74] to improve the quality of SC applications, five different SC applications were redeveloped by adopting a development process inspired by SE. It was observed that the original developers of SC software see the value of applying SE practices and principles for SC, but felt that it was too much work to document the requirements and design. However, this study had limitations because the scientists were not involved in the software development process. To overcome this, it is necessary to engage the scientists in the software development from the start and obtain feedback from them. This would provide an alternate insight in improving the software quality from scientists’ perspective. In this thesis, we plan to accomplish this by conducting another case study where we develop a SC software using approaches from SE, but this time involving the scientists from the beginning of the development process, and then we will obtain their feedback on the use of SE principles and practices.

We began by searching for a suitable project to be the case study for our experiment. We decided on taking up a project in mechanical engineering because of availability and as well as the requirements of the project aligned with our expectations. A brief overview of the case study is given in section 1.1. We completed a literature survey in parallel, to learn the conventional wisdom on applying SE methods to SC software for improving the quality of scientific software. From the results of the survey, we chose the SE approaches that we thought would be suitable for solving this particular SC problem. At that point, we realized that there are various perceptions in applying SE to SC. While applying the SE practices to our case study, we also analyzed the accuracy of those perceptions surrounding the application of SE to SC. In this thesis, we present the feedback from the scientist on applying SE principles and practices to SC. Also, our findings on the accuracy of the common perceptions about applying SE into SC are presented. We strongly believe that the findings of this study will have a significant impact on the attitude towards developing SC software and improving the quality of the same.

1.1 Overview of Case Study

To analyze the common perceptions in applying SE to SC, we needed a scientific or engineering problem that had a significant scope for software development, while still being small enough to develop within a reasonable time frame. We selected Dr. Shankar's research on solidification and casting of alloys in the mechanical engineering department at McMaster University. After a brief discussion with him, we found that there was significant scope for software development in his research for the CleanMag project for Haley Industries.

The software is named SFS (Software for Fraction Solid) and was developed as part of scientific research aimed at reducing the defect ratio by predicting the fraction solid during the solidification of an alloy. The role of SFS in Dr. Shankar's research is to simulate the solidification of the aircraft component being manufactured by a casting process. The main objective of his project is to use the solid fraction output from SFS to predict the quality of the aircraft parts and if necessary, change the parameters of the casting to decrease the defect ratio.

The typical experimental setup for SFS data is shown in figure 3.1. The main elements are the mold containing the molten metal and the water jet under the mold to provide unidirectional heat extraction. The thermocouples are inserted at various locations in the mold to record the temperature. A DAQ (Data Acquisition System) is used for collecting the data from the thermocouples. This data will act as input data for SFS. This temperature versus time data is then processed and used for calculating the solid fraction.

SFS was developed by me (one developer) in Python3 under the guidance of

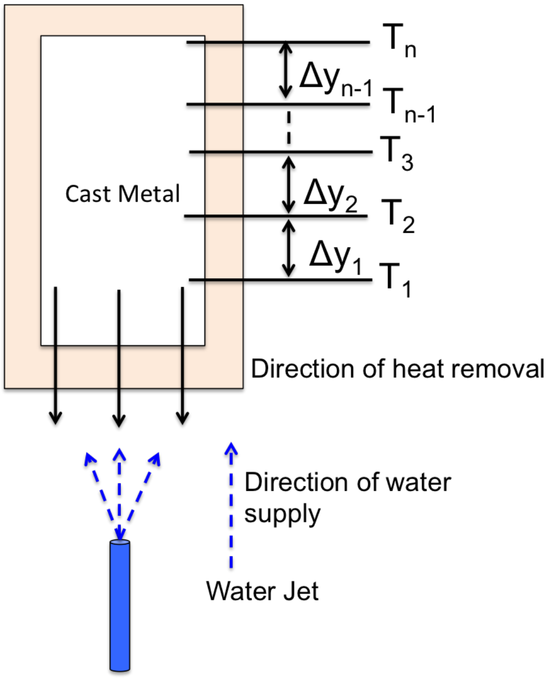


Figure 1.1: Typical experimental setup of SFS

Dr. Smith and consists of several thousand lines of code. When we started to develop SFS, the science was not worked out entirely and was expected to evolve with the software prototype. The backbone of SFS was to predict the temperature throughout the time domain and at all locations between the thermocouples. We used standard numerical methods such as regression, interpolation and ODE solvers from Python to accomplish the task of finding the temperature across the cylinder.

Previously, Dr. Smith had conducted multiple case studies where the existing scientific software was re-developed using SE principles and document-driven approach. He has shown success in improving the quality of the scientific software being redeveloped. However, this study is significantly different from the earlier studies since we developed the software from scratch with the involvement of the scientists from the beginning of the project. Periodic meetings were organized for software and document review, and we had ample communication with the scientists throughout the software development phase.

1.2 Methodology

The primary goal of the methodology is to firm a measurable process to conduct the case study. The results from the process can be used to list the successful and unsuccessful SE approaches of our case study. To accomplish this, the following tasks have been undertaken systematically.

1. Select the SC problem to be our case study and identify a partner to be involved in the software development process.

2. Interview the scientist before the development of the software, with a goal to understand their specialization, coding skills and exposure to SE practices. The details related to pre-development interview is given in section 6.1.1.

3. Analyze the nature of the SC problem and choose the list of suitable SE practices and tools from literature. The details related to the choice of the SE practices and tools is given in section 3.2.1.

4. Develop the software using SE practices and tools identified from the literature. (The SE methods are listed in section 1.3)

5. Involve the scientists throughout the software development process via document reviews and periodic meetings. We also introduced software tools, such as an issue tracker, to facilitate better communication.

6. Interview the scientists after the development of SFS to collect their feedback on the application of SE tools and practices adopted in our case study. We used this information to assess their attitude towards SE for SC. The details related to post-development interview is given in section 6.1.2.

The primary goal in collecting feedback is to identify the SE approaches which work, and those that do not. We also looked for potential modification to improve the adoption of SE in SC going forward.

1.3 Overview of SE Methods

As suggested by Parnas and Clements [57], we followed the “Fake Rational Design approach” to develop SFS. The basic principle behind the “Fake Rational Design approach” is to develop the software from the beginning and prepare the documents as if the software was developed in a rational approach. This method has various advantages such as improved understanding, ease of review etc.

To assist in version control, we used Git, which is a distributed version control system and source code management system. It offers easy branching and merging, which allows one to sandbox features and ideas until they are ready for the mainstream. Git also offers a perfect staging area to organize the changes for each commit. It is fast and inexpensive, which makes it suitable for personal projects too.

The issue tracker in Gitlab was used for communication and task management between people involved in software development. It offers a platform for assigning and reporting bugs and sharing documents related to a specific task. It provides a proper work-flow management through notifications. Each issue can be assigned to an individual, thereby improving accountability. All the issues are located in one shared, central location making them easier to locate.

To conduct document reviews, we used "Task-directed inspection", suggested by Kelly and Shepard [39]. In this approach, every scientist involved in the project was presented with a set of tasks. The difficulty level of the task was kept minimal to make the inspection process easier. The task-list included questions such as "Please read section 5.1 and let us know if the equations used are balanced in terms of units". Each task was listed as an issue and was assigned to the scientists. The scientists had to respond to that issue and close it.

1.4 Scope

This study aims at collecting feedback from the scientists and identifying the successful and unsuccessful SE approaches for our case study. This study has certain limitations discussed below.

- Since the scientists were involved from the start, there is a possibility of bias on their feedback.
- Even though it is a detailed case study, we used only one SC software (SFS) to cover a vast area of research.

1.5 Thesis Outline

This thesis is organized into six (6) chapters. We introduce the essential qualities of a SC software, the role of SE in improving the qualities and the history of SE in SC in chapter 2. In Chapter 3, we discuss the case study in detail with background information about SFS and the list of SE principles and practices applied in developing SFS. In Chapter 4, we present our learnings and observations from applying the SE principles and practices to develop SFS. While developing SFS using SE inspired approaches, we realized that one of the perception around upfront requirements for SC was a myth. We present our arguments in chapter 5 against why we feel it is a myth that upfront requirements are not possible for SC. In chapter 6, we present the feedback of the scientists about SE tools and practices. In chapter 7, we summarize the thesis and discuss the future works in relation to both SFS and SE principles and tools applied to SC. Some of the software artifacts developed during this case study and a guide for developing a SC software based on our experience have been included as appendices.

Chapter 2

Software Quality and Software Engineering

In this chapter, we lay out a foundation for understanding the essential qualities of a SC software and ways to improve it. Firstly, we describe the qualities that are important for any general purpose software. Secondly, we identify and define the qualities which are crucial for SC. Thirdly, we demonstrate the need for SE in SC and finally we present some of the examples where SE is applied to SC.

2.1 Software Qualities

In many cases, the developers of scientific software are 'professional end user developers' who are research scientists working in highly technical, knowledge-rich domains, developing software to further their professional goals [68]. This is due to the complex science behind the software, which the professional software developer, who has only computer science or an SE background, may fail to understand. Typically professional end user developers have little or no education or training in SE [71]. With this group of professional end user developers, especially given the importance of the software they are writing and its societal impact, we need to have clear quality standards.

According to the quality model of ISO 25010 [34], software quality is described as a structured set of eight characteristics namely - functional suitability, performance efficiency, compatibility, reliability, usability, security, maintainability and portability. To facilitate better understanding, the characteristics mentioned above are explained in detail, mostly following the ISO definitions [34].

The figure below summarizes the list of characteristics of the quality of a software product. A detailed definition of all the sub-characteristics of each quality characteristic can be found at [34].

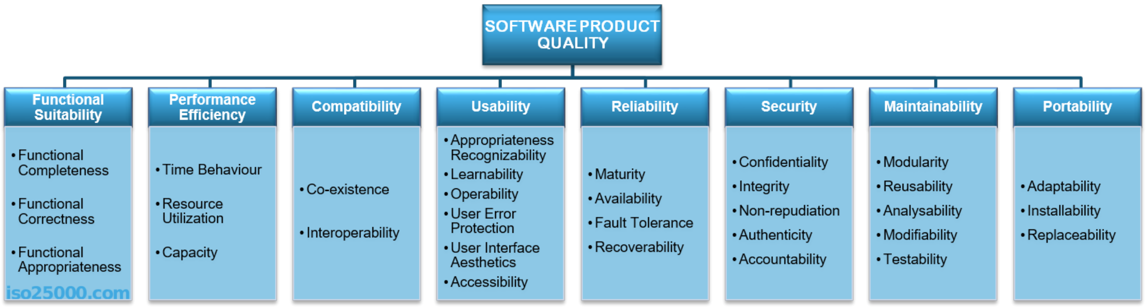


Figure 2.1: Software quality: Characteristics and sub-characteristics [34]

2.1.1 Functional Suitability

Functionality is a characteristic that describes the degree to which the software product satisfies the user's requirement. It can be further characterized into completeness, correctness and appropriateness.

2.1.2 Performance Efficiency

This characteristic emphasizes the efficient utilization of resources. This characteristic is composed of three sub-characteristics namely time, resource utilization and capacity.

2.1.3 Compatibility

The degree to which a product, system or component can exchange information with other products, systems or components, and perform its required functions while sharing the same hardware or software environment. This characteristic is composed of two additional sub-characteristics namely co-existence and interoperability.

2.1.4 Reliability

The degree to which a system, product or component performs specified functions under specified conditions for a specified period. This characteristic is composed of the four sub-characteristics namely maturity, availability, fault tolerance and recoverability.

2.1.5 Usability

This characteristic represents the degree to which a software can be used effectively by the user and achieve satisfaction in the context of usage. This can be further classified into six sub-characteristics: recognizability, learnability, operability, user error protection, user interface aesthetics and accessibility.

2.1.6 Security

The degree to which the software gives access to appropriate data and blocks inappropriate data access. This is achieved with the help of different data authorization levels for different users of the software. This characteristic is composed of five sub-characteristics namely confidentiality, integrity, non-repudiation, accountability and authenticity.

2.1.7 Maintainability

This characteristic represents the degree of effort which the software requires to support for modifications and changes due to corrections and adaptations with respect to change of requirements and environment. This characteristic is composed of five sub-characteristics: modularity, reusability, analysability, modifiability and testability.

2.1.8 Portability

This defines the degree of efficiency with which the software can be transferred between different hardware or different operating environments. This characteristic is composed of the following sub-characteristics: adaptability, installability and replaceability.

2.2 Essential Scientific Software Qualities

The software qualities listed above are not tailored specifically for SC. Hence it is necessary to prioritize the list of qualities that are crucial for SC. In [43], the authors propose that characteristics such as maintainability, portability and reliability are important for a SC application. In addition to the list of qualities mentioned in the article [43], the authors of [75], have identified five (5) additional qualities that are essential: verifiability, validatability, usability, reusability and reproducibility.

Based on the suggestions in the articles mentioned above and the needs of our project, the list of qualities, which we consider essential for SC are maintainability, portability, reliability, verifiability, validatability, usability, reusability and reproducibility. Several of these are already defined in sections 2.1.7, 2.1.8, 2.1.4 and 2.1.5. The remaining qualities (verifiability, validatability, reusability and reproducibility) are defined below.

2.2.1 Verification and Validation

In a SC context, verification means “solving the equations right” and validation is “solving the right equation” [75]. The degree of efficiency and effectiveness with which verification and validation can be carried out are called verifiability and validatability.

2.2.2 Reproducibility

Reproducibility is the ability of the software to produce identical results when the code is rerun in the future, possibly through an independent third party [14].

2.2.3 Reusability

Reusability is the degree to which existing assets such as code, test suites, designs and documentation can be used in some form in the creation of software systems rather than building software systems from scratch [45].

2.3 The Need for SE in SC

In the past, many attempts have been made to improve the quality of SC applications because of their huge societal impact. SE, from its end has offered tools and techniques to improve the quality of a SC software. However, they are not used properly because learning and employing SE principles and practices has a nontrivial learning curve and most of the SC applications are developed by end user developers [68] who do not have any SE background. In addition to this, application of SE principles and practices demands an upfront effort to create the software artifacts and some people believe that structured SE methods are not worth the effort [84].

In a case study conducted by Carver et al [10], the author mentioned that “scientists do not view rigid, process-heavy approaches, favorably and prefer agile methodologies”. This means that some software development approaches, such as a document-driven approach, which are plan based, are not preferred by end user developers. As an example from a scientific software developer, Roache [63, p. 373] considers reports for each stage of software development as counterproductive. Another argument against using a rational process, where software is derived from precise requirements, centers around the opinion that, in science, requirements are impossible to determine upfront, because the details can only emerge as the work progresses [10, 17, 35, 69, 71].

Even though end user developers do not view structured SE approaches favorably, there is evidence in the literature that these approaches improve the quality of SC applications. For instance, the article [79] shows that the quality of statistical software for psychology is generally improved when developers use the structured CRAN (Comprehensive R Archive Network) process and tools, versus an ad hoc process. A case study described in the article [75], highlights the value of proper documentation by redeveloping nuclear safety analysis software. Twenty seven (27) worrisome documentation problems were found, including incompleteness, ambiguity, inconsistency, verifiability, modifiability, traceability and a lack of abstraction. Emphasizing the importance of documentation in the development of scientific software, Chilana et al. [11], Dubey et al. [16] and Fangohr et al. [18] argue that the developers of SC software keep changing throughout the project, as people join and leave the group, making documentation crucial for SC software. A redevelopment experiment with five existing projects [74] enabled the code owners (as ascertained through interviews) to clearly see the value of documentation. However, mirroring other studies [10], the code owners felt documentation takes too much time.

In article [19], the author observes "growing concern about the reliability of scientific results based on ... software". Similarly, embarrassing failures have occurred, like a retraction of derived molecular protein structures [52], false reproduction of sonoluminescent fusion [60], and fixing and then reintroducing the same error in a large code base three times over 20 years [51]. Typical SC software has a long lifetime [61] and long-term maintenance of a SC software is prohibitively expensive [1]. It is evident that there is a growing need to concentrate on maintainability and reusability of SC software.

In an experiment run by the Mozilla Science Lab, software engineers have reviewed selected pieces of code from published papers in computational biology with the aim to improve the quality of researcher-built software. According to Kaitlin Thaney, the Lab director of Mozilla [25], "Scientific code does not have that comprehensive, off-the-shelf nature that we want to be associated with the way science is published and presented". This clearly highlights that there are problems with respect to reproducibility of published SC applications. A similar issue is highlighted in a recent study of the code for 402 computer systems papers where only 48.3% of the code was both available and compilable [12]. Although reproducibility is the cornerstone of the scientific method, until recently it has not been treated seriously in software [7]. Reproducibility problems are even more extreme when the goal is replicability. A third party should be able to repeat a study using only the description of the methodology from a published article, with no access to the original code or computing environment [7]. However, replicability is rarely achieved, as shown for microarray gene expression [32] and for economics modelling [33]. In the recent years multiple conferences, workshops and individuals are calling for dramatic change [6] in addressing the problem of reproducibility. For instance, in the article [59], the author claims that the source code must be published along with scientific publications to improve the reproducibility of the software.

A recent report on directions for SC software research and education states: "While the volume and complexity of [SC software] have grown substantially in recent decades, [SC software] traditionally has not received the focused attention it so desperately needs ... to fulfill this key role as a cornerstone of long-term collaboration and scientific progress" [65]. The community of SC software is finally realizing that current practices are not enough to improve the quality of SC applications and that their software development approach needs to be reformed.

To encourage the end user developers to learn and apply SE practices and principles, Wilson [90] and Messina [50], organized a workshop to train the end user developers in necessary software skills so that they can use SE methods while developing SC software to improve the quality of SC applications. Educating the scientists proved to be a successful approach. A recent proposal on future directions for SC software research and education [65] recognizes the desperate need for change - incorporating SE in SC. The leaders of SC software recognize that an interdisciplinary approach provides the path forward. They believe that the solution to improve the quality of SC software is applying, adapting and developing SE methods, tools and techniques. However, typical software processes are a barrier to progress. "To break the gridlock, we must establish a degree of cooperation and collaboration with the [SE] community that does not yet exist" [19]. "There is a need to improve the transfer of existing practices and tools from ... [SE] to scientific programming. In addition, ... there is a need for research to specifically develop methods and tools that are tailored to the domain" [82].

2.4 Examples of SE Applied to SC

SE is defined as "an engineering discipline that is concerned with all aspects of software production" [81]. Other notable definition for SE is given by IEEE [86] where SE is defined as "the systematic application of scientific and technological knowledge, methods and experience to the design, implementation, testing, and documentation of software". The primary goal of SE is to build software while minimizing defects, complexity and managing qualities like verifiability, usability, maintainability, availability, reliability and other quality attributes of the software.

SC applications are usually developed by scientists who are guided by the impetus to prove a scientific hypothesis. Scientists value the software according to its progress in scientific research and do not typically validate the software until an error affects the accuracy of results [70]. The authors of [61, 64] suggest that the quality of SC software can be improved by the use of standardized SE practices. The following are a few key examples from literature which demonstrate the need for SE principles and practices to improve the quality of SC software.

- According to Greg Wilson [56], Many scientists and engineers spend a lot of time in writing, debugging, and maintaining software; but only a handful have been taught how to do this effectively: Usually, after a couple of introductory courses, they are left to rediscover (or reinvent) the rest of programming on their own. This resulted in them having very little idea on the reliability and accuracy of the programs written by them.

- There is a common perception among scientists that requirements can be discussed verbally rather than documenting properly [69] due to which testing becomes difficult as there is no written information available to verify the fulfillment of the requirements [69, 70].

- According to Post and Kendall [61], the development of SC software requires good project management, risk identification, software quality engineering, validation, and verification.

- In [27], the authors describe an approach to simplify the development of scientific applications by designing tools, applying domain engineering concepts, and using domain-specific modelling, which are modern software engineering methods for automating software development.

- In [44], the authors use software quality models as a tool for the quantitative assessment of attributes that affects the quality at each stage of software development.

- In the article [28], the authors state that automation of tasks was followed in the Trilinois project to improve maintainability.

The above references from the literature makes it clear that adopting SE principles and practices is a way to improve the quality of the SC applications. This is also emphasized in [44], where the author has discussed a detailed literature summary of related works in this field.

Chapter 3

Case Study in Detail

In this chapter, we present the details about the case study. This chapter is organized into two broad sections. In the first section, we will see the details about the project and in the second section, we will discuss the SE practices applied.

3.1 SFS in Detail

Haley Industries is a Canadian company that manufactures lightweight metal castings for use in aerospace and specialized engineering applications [85]. Dr. Sumanth Shankar, a professor in the Department of Mechanical Engineering at McMaster University is working on a project called "CleanMag (Magnesium Cleaning)" to decrease the defect ratio of the aircraft parts manufactured by Haley Industries. SFS is part of the CleanMag project. SFS predicts the nature of solidification in the castings based on the operating conditions before the actual casting takes place. The input data for the software was obtained from the experiments and trials conducted at both McMaster University and at Haley Industries.

The experiment was devised with a cylinder made of a sand mould with a cavity whose dimension ensure approximately unidirectional heat removal. This experiment helps in understanding the solidification process. The liquid metal alloy is poured into the sand cavity. During solidification, heat is given out from the liquid phase to form a solid. During this process, the alloy undergoes a phase transition from liquid to 2 phase (solid + liquid) and finally to solid. In this project, software is developed to estimate the fraction of solid present in the 2 phase zone. This data can then be used to characterize the solid fraction as a function of temperature and rate of cooling. The typical experimental setup is shown in Figure 3.1.

The experimental setup has the following components:

1. Sand mould: This acts as the container into which the metal alloy is poured.

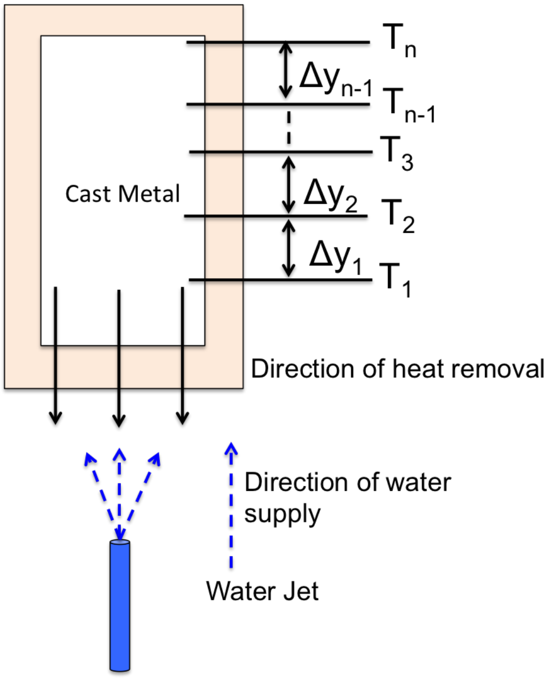


Figure 3.1: Experimental setup for cooling a liquid metal alloy

2. Thermocouples: These act like thermometers that record the temperature with respect to time. Many thermocouples are placed across the cylinder at different locations. They are fixed inside the sand mould so that they have contact with the metal alloy to avoid any error during temperature measurement. They are represented in Figure 3.1 by the label 'T_i', where i is the thermocouple number.

3. Water jet: The water jet is present at the bottom of the cylinder to aid in the cooling process and to facilitate $1D$ heat transfer.

4. Cavity: There is a cavity in the sand mould into which the molten metal alloy is poured.

Typical representation of data for an alloy from a single thermocouple is shown in Figure 3.3. When the molten alloy is poured into the cavity, the temperature of the thermocouple begins to raise from room temperature to the actual temperature of the alloy as shown in the Figure 3.2.

For computation of fraction solid, we consider the data only from the maximum temperature and remove the extraneous data that accompanies the filling of the sand mold. Therefore, the data begins at the maximum temperature. It consists of three

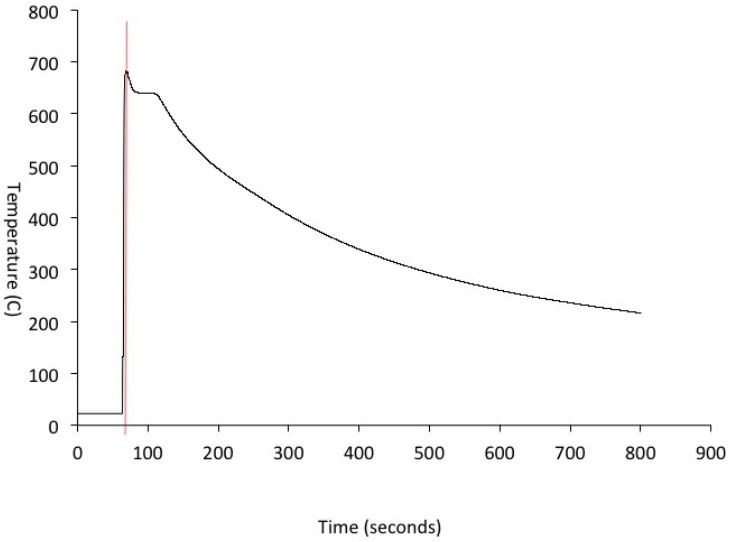


Figure 3.2: Typical data from a thermocouple including the extraneous data from the time of pouring the alloy

different zones, namely solid zone, liquid zone and 2 phase zone (solid + liquid), and important points in the data signifying solidification phenomena. Understanding solidification involves identifying the points at which the elements present in the alloy start and finish solidifying. These points are called liquidus, eutectic and solidus points and are marked in the zoomed view of the data as shown in Figure 3.3. The liquidus point occurs at the temperature where solids first start to form. It is represented by a sudden change in curvature of the cooling curve. A eutectic point is when the rate of energy leaving the material matches the rate of energy entering through solidification. At the eutectic point the temperature remains constant until all of the material has solidified. The solidus temperature specifies the temperature below which a material is completely solid.

Before the start of solidification, marked by liquidus point, the fraction of solid is zero and after the end of two phase zone marked by solidus point, the fraction solid is 1. The fraction solid is obtained by solving the following ODE:

$$\dot{f}_s(f_s,t) = \frac{C_v(f_s)}{L\rho(f_s)}\left[\frac{\partial T(t)}{\partial t} - \alpha(f_s)\frac{\partial^2 T(t)}{\partial y^2}\right]$$

L is the specific latent heat for a particular substance ($\mathrm{J\,kg^{-1}}$).

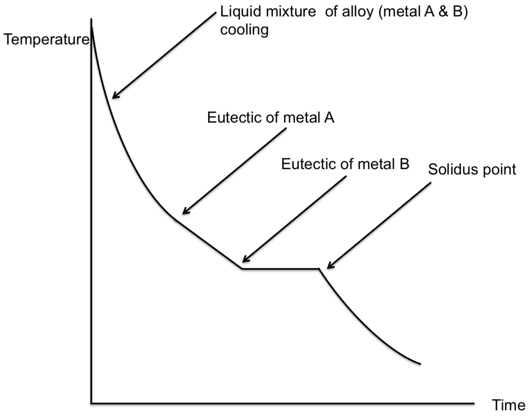


Figure 3.3: Typical cooling in a binary alloy with liquidus, solidus and eutectic points identified ([46])

$C_v(f_s)$ is the volumetric heat capacity in the two phase zone (J/(m^3 °C)).

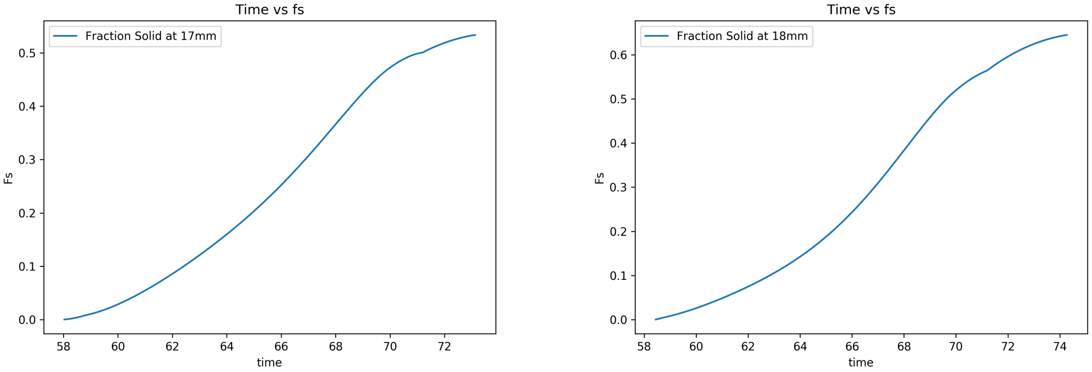
$\alpha(f_s)$ is the thermal diffusivity in the two phase zone (m^2/s).

$\rho(f_s)$ is the density of the alloy in the two phase zone (kg m^{-3}).

$T(y,t)$ is the function which returns temperature at any time t and location y, from which $\frac{\partial T}{\partial t}$ and $\frac{\partial^2 T}{\partial y^2}$ can be derived, as required.
The derivation of this equation is provided in the requirements documentation for SFS. The requirements will be discussed further in section 3.2.3.1. A typical output from SFS for the fraction solid with respect to time are shown in Figure 3.4. The starting time in the plot (58 seconds) represents the liquidus point which is the time at which solidification starts ($f_s = 0$ before liquidus point). The end time (72 seconds) represents the solidus point which marks the end of two phase zone, beyond which fraction solid is 1. The solid fraction does not reach 1 because of some issues with the calibration of the sensors. Since the emphasis of this thesis is on SE, not the science of solidification, the details of the input data will not be explored further here.

3.2 SE Practices and Tools in Detail

In this section, we describe some of the software engineering practices applied during the development of SFS and discuss the reason for selecting those SE practices and tools.



(a) Fs vs time (seconds) at 17mm

(b) Fs vs time (seconds) at 18mm

Figure 3.4: Typical output from SFS

3.2.1 Selection of SE Practices and Tools

Here we describe the nature of SC problem related to this case study and provide reason for choosing some of the critical SE practices and tools used for developing SFS. We collected ample information on the SC problem during initial discussion and the pre-development interview. Based on this information and through an extensive literature survey, following SE practices have been selected for our case study.

1. **Design for change:** The science and the mathematics behind the software was still in the exploratory phase. Hence, changes are inevitable at any point during the development phase. To support for changes during the development, we adopted "Design for change", which is based on designing a software anticipating changes. This is explained in detail in section 3.2.5.

2. One of the primary requirements of SFS was to fit the temperature versus time data and various approaches were suggested to fit the data. Hence, it was obvious that trial and error experiments will be necessary to choose the most suitable method. To accommodate the trial and error experiments, we decided to use "Regression testing". It provides a way to evaluate different numerical techniques used in the trial and error experiments. This is explained in section 3.3.

3. SFS will be used for a longer period of time and may be interfaced with existing software to automate the input of material properties. Hence, maintainability is critical for SFSand we adopted a "Document driven design" to support for the maintainability requirement of SFS. The details of this approach is discussed in

section 3.2.3. However, we combined it with “Faked rational process design” because the mathematics and the scientific aspects of the software were yet to be finalized. Therefore, we presented the documents as if we followed a typical document driven design, even though we accommodated changes when necessary. This is explained in section 3.2.4.

In addition to the above mentioned SE practices, we also used other SE practices and tools which is explained in detail in section 3.2.2.

3.2.2 SE Practices Applied to SFS

The following SE practices and tools were applied during the development of SFS.

1. Document Driven Design
2. Faked rational process design
3. Design for change
4. Regression testing
5. Task based inspection
6. Issue tracking
7. Git

3.2.3 Document Driven Design

Document-driven design is a software development methodology in which a document is produced at every phase such as requirement analysis, design, implementation, testing and maintenance of the software development cycle. This document acts as the driving force for the next stage. The document produced at each phase serves as a means to collaborate with the project team members.

In this case study, we followed a document driven approach, which means that every stage of the software development has a document associated with it. Each of these documents represents each stage in an ideal waterfall model. The list of documents we produced during the development of SFS includes SRS (Software Requirement Specification), MG (Module Guide), MIS (Module Interface Specification) and the code. The main purpose of each document along with suitable examples from the case study are presented below.

3.2.3.1 Software Requirement Specification (SRS)

This is the first step in the software development process. The aim is collecting the set of functionalities or requirements that the software must accomplish. The SRS is a document that clearly and precisely describes each of the essential requirements (functions, performance, constraints and quality attributes) of the software and external interfaces [31]. This step is important because many developers consider that the SRS helps in improving the qualities [80] mentioned in section 2.1. As an example, we present the table of contents in Figure 3.5 and discuss the important sections of the SRS which we developed for this case study.

To create the SRS for SFS, we followed a standard template suggested in [77, 78]. The SRS has the following sections: reference material, introduction, background, general system description, specific system description, requirements, likely and unlikely changes. The contents of each section are explained briefly.

Reference material

This is the first section of the SRS and it records information for ease of reference. The information contained in this section includes the table of units, table of symbols and the list of abbreviations and acronyms used in the SRS.

Introduction

This section gives a brief introduction to the scientific problem being discussed. It describes the primary purpose of the SRS document, its scope and the intended readers. This section also provides a road map to the organization of the SRS.

Background

This section provides a brief explanation about the scientific information necessary to understand the problem being discussed. The aim is to provide the readers of the SRS with essential background information for the problem under consideration.

General System Description

In this section we describe the general information about the system, identify the interfaces between the system and its environment, and describe the user characteristics and the system constraints. Some of the information presented in section 3.1 is also a part of this.

Specific System Description

This section presents the problem description, which gives a high-level view of the problem to be solved. This is followed by the solution characteristics specification, which presents the assumptions, theories, definitions and finally the

1. Reference Material: a) Table of Units b) Table of Symbols c) Abbreviations and Acronyms
2. Introduction: a) Purpose of Document b) Scope of Requirements c) Intended Audience d) Organization of Document
3. Background
4. General System Description: a) System Context b) User Characteristics c) System Constraints
5. Specific System Description:
   a) Problem Description: i) Terminology and Definitions ii) Physical System Description iii) Goal Statements

   b) Solution Characteristics Specification: i) Assumptions ii) Theoretical Models iii) General Definitions iv) Data Definitions v) Instance Models vi) Data Constraints vii) Properties of a Correct Solution
6. Requirements:
   a) Functional Requirements: i) Configuration Mode ii) Calibration Mode iii) Calculation Mode
   b) Non-Functional Requirements: i) Look and Feel Requirements ii) Usability and Humanity Requirements iii) Installability Requirements iv) Performance Requirements v) Operating and Environmental Requirements vi) Maintainability and Support Requirements vii) Security Requirements viii) Cultural Requirements ix) Compliance Requirements
7. Likely Changes
8. Unlikely Changes
9. Supporting Information

Figure 3.5: Table of Contents for SRS

instance models that models the scientific problem being solved. Some of the important contents of this section are the goal statements, data definitions and the instance models. For instance, the goal statement (G1) from our SRS is presented below.

> For a given experiment with a metal alloy, using the thermocouple locations, temperature readings, material properties and initial conditions,
>
> G1: SFS computes the solid fraction (f_s) as a function of temperature and cooling rate ($f_s(T, \frac{dT}{dt})$).

Other parts of this section such as data definitions, instance models etc are discussed throughout this thesis.

Requirements

This section provides the functional requirements, the business tasks that the software is expected to complete, and the nonfunctional requirements, the qualities that the software is expected to exhibit. Some of the requirements of the SFS are presented in section 5.1 of this thesis and readers are encouraged to read section 5.1 to get a feel of the functional requirements of SFS.

Likely and Unlikely changes

This section records the information about possible changes in SFS. This is essential for creating a good design of the software which is discussed in detail in section 3.2.5. Likely changes are anticipated and the software is designed to support them to improve maintainability. The unlikely changes are considered to be so significant that they would fundamentally change SFS. If these changes become necessary, significant changes will be necessary for the SRS, subsequent documents and the code. Some of the anticipated changes are presented in 3.2.5.

The SRS of SFS is available as part of appendix to this thesis (B). Readers are encouraged to read the SRS for a complete overview of the software discussed in this project.

3.2.3.2 Module Guide (MG)

This is a document produced during the design of the software. Software design involves decomposing the software into modules, where the definition of a module adopted here is a “work assignment” [55]. The Module Guide (MG) gives the summary of what the modules are intended to do and the relationship between them [80].

The primary purpose of developing the MG is to improve the maintainability of the software. This is based on the principle of ‘abstraction’ which allows both designers and maintainers of the SFS to easily identify the parts of the software that they want to consider for modifications or additions without needing to know the unnecessary complex details.

The design of the MG follows the rules layed out by [58], as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used in only one module.
- Any other program that requires information stored in a module’s data structures must obtain it by calling access programs belonging to that module.

The module decomposition of SFS is presented in Table 3.1 to give an overview of the modules of SFS contained in the MG document.

Each module has four (4) fields. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. If the entry is *Python*, this means that the module is provided by Python. *SFS* means the module will be implemented by the SFS software. As an example, one of the modules from the MG of SFS is presented below.

Parameter Specification Module ()

Secrets: Constants used by SFS.

Services: Stores the parameters needed for the program, including material properties, processing conditions and numerical parameters. The values can be read as needed. This module knows how many parameters it stores.

Implemented By: SFS

The module information about the other important modules of SFS are discussed in section 3.2.5.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Control Input Configuration Experiment Parameter Specification Temperature Identify Points Calculation Load Output Verification Output
Software Decision Module	Interpolation Piecewise Data Structure Sequence Services Interpolation Linear Regression ODE Solver Plot

Table 3.1: Module Hierarchy

3.2.3.3 Module Interface Specification (MIS)

This is another design document that we produced as part of a document driven design. In the previous section (3.2.3.2), we presented the MG, which lists the modules in SFS. However, the information presented in the MG is not enough for each module to be implemented independently. The syntax and semantics of the access routines for each module are still needed [80]. The MIS aims at describing the syntax and semantics of each module listed in MG, but does not mention how they will be implemented. The structure of the MIS is adapted from the template proposed in [29] and ideas from [57], with the addition that template modules have been adapted from [21]. The MIS consists of the following sections:

- **Syntax**: The syntax of the MIS for each module documents the imported and exported data types and the exported access programs. These access programs act as the interface, which remains stable with any changes in the internal design.

- **Semantics**: The semantics of the MIS for each module gives the information about how the state of the module and its output will change depending on the current state and the provided input.
- **State Variables**: The state variables of each module is used to give a memory to the module. Some of the modules also have state invariant section, which represents the property that holds true, at all time, for the state variable under consideration.
- **Access Program Semantics**: This section represents the semantics of each access program present in the module. It contains three (3) sections namely: Output, Transition and Exception.
- **Assumptions**: This section lists the assumptions governing the use of the module. For instance, "init() must be called before any other access program is used".

The MIS of the module presented as example in section 3.2.3.2 is presented here to ensure continuity. The MIS of other more interesting modules are presented in section 3.2.5.

MIS of Parameter Specification Module

Module

specParam

Uses

None

Syntax

Exported constants

#From the Table 6 in SRS
$H_{\text{min}} := 0.001$
$H_{\text{max}} := 100$
$D_{\text{min}} := 0.001$
$D_{\text{max}} := 100$
$\vdots$
$\vdots$
$C_v^S{}_{\text{min}} := 1 \times 10^{-4}$
$C_v^S{}_{\text{max}} := 1 \times 10^{4}$

Assumptions

None

Access Routine Semantics

N/A

3.2.3.4 Code

This is the section where the software is implemented. The system implementation is the transformation of the design to a work product [80]. The design decisions and algorithms are finalized only during this stage. SFS is implemented in Python3. We also used version control systems and issue tracker to help us during the coding phase. These tools are discussed in section 3.5.

Code for few of the modules of SFS is given in appendix (D).

3.2.3.5 Software Testing

In this section, the software developed is tested to improve the confidence in the code. An important purpose of testing in SC is to describe the quality of the software [80]. A document driven approach improves the verifiability and validatability of the software as it provides the list of requirements and qualities important for the software. The software is tested for the fulfillment of the functional and non-functional requirements specified in the SRS. Sometimes, a detailed test plan is also developed to verify the quality of the test cases and ensure coverage of the tests. However, we did not develop a test plan for the verification of SFS. We validated our equations through document reviews and feedback during meetings and presentations.

We tested the important modules of SFS using pytest, the system test framework available in Python. SFS deals with experimental data, thus, the exact solution is not known. Hence we tested the modules by choosing simpler test cases with known solutions. For instance, to test the temperature module, which has access programs to compute the gradients of the temperature, we chose an analytical function which resembled our temperature data, computed the gradients using the software and compared it against the gradients obtained by differentiating the analytical function. The test results were good and the maximum relative error was less than 0.03%, which was acceptable for the SC problem under consideration.

Testing also plays a crucial role in SC because trial and error approaches are inevitable in SC. For instance, SFS was subjected to different numerical trial and error experiments. Regression testing, defined in section 3.3 was followed to conduct meaningful trial and error experiments. Even though, we did not follow regression testing since the beginning, we benefited greatly from it to quantify and conclude the trial and error experiments.

3.2.4 Faked Rational Design Process

A rational process, also called waterfall model, is a software development approach in which the development starts with the requirements, then moves to design where the developer creates the modules and decides on the algorithm with the help of different SE concepts such as “Separation of concerns,” “Information hiding” etc. The system architecture design phase is succeeded by interface specification, during which any major design decisions are documented. Finally, the implementation and maintenance phase completes the entire development process. This rational software development process is considered “not practical” because according to literature [35, 10, 71], it may not always be possible to specify the requirements at the beginning of the process. Sometimes, new constraints may emerge later, which makes the application of rational software development process not suitable for SC.

Parnas and Clements [57] suggested the idea of a “Faked rational process design” in 1986 to overcome the shortcomings of a “Rational process design.” According to this method, the most logical way to present the documentation is to fake it as if the software development followed a rational process design.

In our case study, we were able to write the “Requirements specification” at the beginning of the project, which is explained in detail in chapter 5 of this thesis, but we still needed to start the implementation phase simultaneously for the deliverables of the weekly meetings. In addition to the meeting deliverables discussed above, we also knew that this software would be used for a long time by different people, so we wanted to avoid the “Mythical Man Month effect” [20], which means that when new people join the project they should not have to depend on the developer for information about the software. One of the significant advantages of this approach to documentation is the amelioration of the Mythical Man Month effect. Hence, we adopted this approach of faking the rational process design. Besides providing quality improvements and quality assurance, this method also provides documentation that is easier to reuse and maintain as the documentation is understandable and standardized [9]. In addition to this, the main advantage of a rational process, which closely parallels how engineers typically think about their workflow can be obtained by documenting the work products as if they were developed and written following the waterfall model [57].

3.2.5 Design for Change

This is a practice followed in SE, in which existing code is treated as an asset. A change in technology, science or society may warrant a change in the software. Hence it is essential to keep changes in mind while designing the software. Some of the change types are mentioned below.

1. Corrective: This type of change involves fixing the bugs in the software code.
2. Adaptive: This type of change deals with changing the software to adapt to new technology such as a new hardware or software platform.
3. Perfective: This change type aims at adding new functionalities to improve the performance.

In our case study, the software to be developed needed to accommodate changes because we were not sure about the exact mathematics which will be used to solve for fraction solid. In other words, it was evident that the software will be subjected to “Trial and error” methods before agreeing on a specific design. In addition to this, the software was planned to be interfaced with other scientific software for input data such as material properties of the alloy being cast. Hence, it was necessary to design the software in such a way that it can accommodate these changes. These requirements led us to adopt the approach of "Design for Change", which means to identify all the possible changes and design the software to accommodate these changes when the need arises. For instance, without "Design for Change", a new version upgrade in a scientific software to which SFS is linked could make it useless. We identified the following anticipated changes in SFS, which are the sources of the information that are to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. Anticipated changes are labelled by **AC** followed by a number.

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.

AC6: The algorithm used to fit the temperature data may change.

AC7: The constraints on the output results.

AC8: How the governing ODEs are defined using the input parameters.

AC9: How the overall control of the calculations is orchestrated.

AC10: The implementation for the sequence (array) data structure.

AC11: The algorithm used for the ODE solver.

AC12: The implementation of plotting data.

While designing SFS, each of these anticipated changes is encapsulated in a module. For instance, at the beginning of the project, we knew that we need temperature and its derivatives at any location y across the cylinder and at any time t till the end of input time. This means that given the input data, which is the "Temperature versus time" from each thermocouple, we need a function, represented by $T(y,t)$ which can give us temperature at any time 't' and at any location 'y', inside the cylinder including locations at which there is no thermocouples and time at which we do not have temperature data.

Hence, we decided to use fitting functions to the input data to find the temperature across the time and location domain. We identified that the type of fitting function would be a likely change in this software. Hence we encapsulated the fitting function of input data into a module called “Piecewise module” and obtaining temperature and its derivatives into another module called “Temperature module.” The detailed design of the Piecewise module and Temperature module is given below.

3.2.5.1 Piecewise Module

The primary purpose of this module is to fit the input data, which is the “Temperature versus time” from each thermocouple. Each thermocouple has its own “Temperature versus time” which had to be fit to obtain temperature at times which is not a part of the input data. We had different options of fitting line regression, spline, interpolation etc. Our design for Piecewise module had to accommodate these changes.

We started with regression and tried to fit one polynomial to each thermocouple. However, this approach failed because of the following two reasons:

1. The fit obtained was not the best at regions of interest, like at the liquidus and eutectic points.
2. At higher locations (greater location values), polynomial fit had oscillations which further decreased the quality of the fit.

Hence, it was evident that one polynomial was not enough to fit the entire data. Therefore, we decided to use piecewise polynomials to fit the data. Based on the nature of the data, we decided that three sections would suffice for the data handled by SFS. We decided to fit the data using three polynomials for each thermocouples. The detailed description of the piecewise regression and the initial guess value for the fit is described in the below section, which is followed by the module design of Piecewise module.

Derivation of Piecewise Curve Fitting

The equation for the three piece regression is as follows:
$p_1(t) = a_1t^3 + b_1t^2 + c_1t + d_1$ (between t_0 and t_1)
$p_2(t) = a_2(t-t_1)^3 + b_2(t-t_1)^2 + c_1(t-t_1) + d_2$ (between t_1 and t_2)
$p_3(t) = a_3(t-t_2)^6 + b_3(t-t_2)^5 + c_3(t-t_2)^4 + d_3(t-t_2)^3 + e_3(t-t_2)^2 + f_3(t-t_2) + g_3$ (between t_2 and t_4)
Substituting $t = 0$ in $p_1(t)$,
$p_1(0) = T_0 = a_10^3 + b_10^2 + c_10 + d_1$
Hence, $d_1 = T_0$

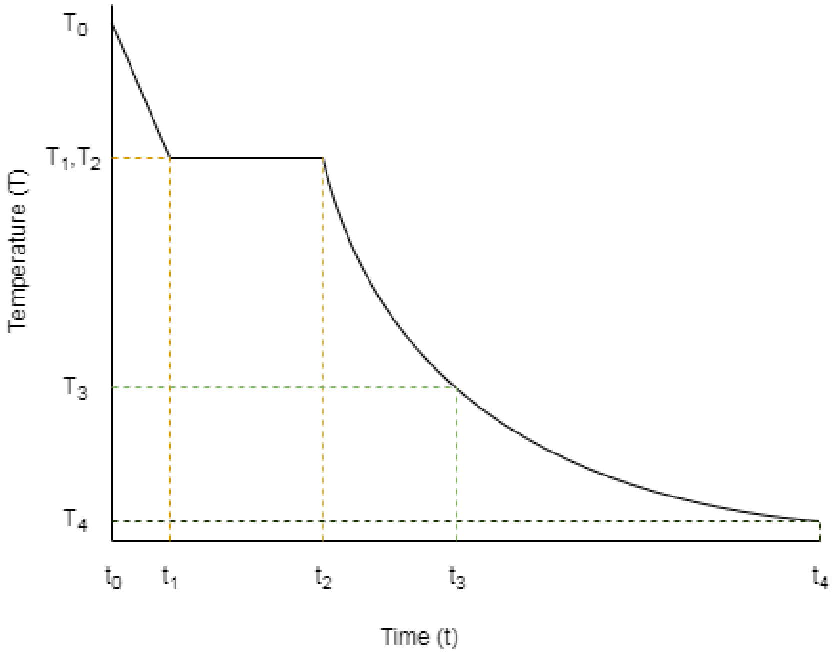


Figure 3.6: Piecewise regression breakdown of a thermocouple

$p_1(t_1) = p_2(t_1)$ for continuity.
So $d_2 = p_1(t_1) = a_1{t_1}^3 + b_1{t_1}^2 + c_1t_1 + d_1$
Similarly, $p_2(t_2) = p_3(t_2)$ for continuity.
So, $g_3 = p_2(t_2) = a_2(t_2 - t_1)^3 + b_2(t_2 - t_1)^2 + c_1(t_2 - t_1) + d_2$
This idea was generalized for higher order polynomials.

Initial Piecewise Regression Equation Guesses The initial guess for the coefficients is obtained from the derivation as shown below. The value of initial guess is crucial for finding the fit. When the initial guess is closer to the actual coefficients, the probability of finding the fit is high.

Guess between t_0 to t_1 Assuming a linear interpolation for the guess,
$p_1(t) = c_1t + d_1$ where $p_1(t_0) = T_0$
substituting $d_1 = T_0$, $p_1(t) = c_1t + T_0$
$p_1(t_1) = T_1$ so, $c_1t_1 + T_0 = T_1$
$c_1 = \dfrac{T_1 - T_0}{t_1}$
$\therefore$, $p_1(t) = \dfrac{T_1 - T_0}{t_1}t + T_0$

Guess between t_1 to t_2 Assuming a linear interpolation for the guess,

$$p_2(t) = c_2(t - t_1) + d_2 \text{ where } d_2 = p_1(t_1) = c_2(t - t_1) + p_1(t_1)$$
$$p_2(t_2) = T_2 \text{ so } c_2(t_2 - t_1) + p_1(t_1) = T_2$$
$$c_2 = \frac{T_2 - p_1(t_1)}{t_2 - t_1}$$
$$\therefore p_2(t) = \frac{T_2 - p_1(t_1)}{t_2 - t_1}(t - t_1) + p_1(t_1)$$

Guess between t_2 to t_4 For this section, we needed to add curvature. So, we used a quadratic interpolation for the guess.

$$p_3(t) = e_3(t - t_2)^2 + f_3(t - t_2) + g_3$$
$$p_3(t_2) = T_2 = p_2(t_2) = g_3$$
$$p_3(t_3) = T_3 = e_3(t_3 - t_2)^2 + f_3(t_3 - t_2) + g_3$$
$$p_3(t_4) = T_4 = e_3(t_4 - t_2)^2 + f_3(t_4 - t_2) + g_3$$

Solving for e_3, f_3 with the given matrix:

$$\begin{bmatrix} (t_3 - t_2)^2 & (t_3 - t_2) \\ (t_4 - t_2)^2 & (t_4 - t_2) \end{bmatrix} \begin{bmatrix} e_3 \\ f_3 \end{bmatrix} = \begin{bmatrix} (T_3 - f_3) \\ (T_4 - f_3) \end{bmatrix} = \begin{bmatrix} (T_3 - T_2) \\ (T_4 - T_2) \end{bmatrix}$$

Using Cramer's Rule $Ax = b$:

$$A = \begin{vmatrix} (t_3 - t_2)^2 & (t_3 - t_2) \\ (t_4 - t_2)^2 & (t_4 - t_2) \end{vmatrix} = (t_3 - t_2)^2(t_4 - t_2) - (t_4 - t_2)^2(t_3 - t_2)$$

$$B = \begin{vmatrix} T_3 - T_2 & (t_3 - t_2) \\ T_4 - T_2 & (t_3 - t_2) \end{vmatrix} = (T_3 - T_2)(t_4 - t_2) - (T_4 - T_2)(t_3 - t_2)$$

$$C = \begin{vmatrix} (t_3 - t_2)^2 & (T_3 - T_2) \\ (t_4 - t_2)^2 & (T_4 - T_2) \end{vmatrix} = (t_3 - t_2)^2(T_4 - T_2) - (t_4 - t_2)^2(T_3 - T_2)$$

Applying Cramer's rule to find e_3

$$e_3 = \frac{B}{A} = \frac{(T_3 - T_2)(t_4 - t_2) - (T_4 - T_2)(t_3 - t_2)}{(t_3 - t_2)^2(t_4 - t_2) - (t_4 - t_2)^2(t_3 - t_2)}$$

Finding f_3

$$f_3 = \frac{C}{A} = \frac{(t_3 - t_2)^2(T_4 - T_2) - (t_4 - t_2)^2(T_3 - T_2)}{(t_3 - t_2)^2(t_4 - t_2) - (t_4 - t_2)^2(t_3 - t_2)}$$

With this initial guess values, the curve_fit method in scipy module can be used to determine the values of the coefficients.

3.2.5.1.1 MG of Piecewise Data Structure Module

Secrets: The data structure used to store the thermocouple data.

Services: This module provides all the necessary information regarding a thermocouple like - coefficients of the equation for all 3 sections used to construct the temperature values, the break points used for fitting, the phase change points and a boolean value which tells whether a fit was obtained for the data.

Implemented By: SFS

In Piecewise module, the secret is the data structure used to store the thermocouple data. This means the data structure may vary in the future depending on the data type or fitting technique used but the service it provides does not change. For example, if instead of 3 section piecewise polynomial fitting, if we needed '*n*' sections or if we use a different curve fitting technique such as 'Smoothing', then the data structure which is the secret of this module alone will change. This is explained better with the "Module Interface Specification" as shown below.

3.2.5.1.2 MIS of Piecewise Data Structure Module

Template Module: PiecewiseADT

Uses: Regression, SeqService

Syntax:

Exported Types: PiecewiseT = ?

Exported Access Programs

Name	**In**	**Out**	**Exceptions**
PiecewiseT	$x:\mathbb{R}^n, y:\mathbb{R}^n$, $x_1^{\text{init}}:\mathbb{R}$, $x_2^{\text{init}}:\mathbb{R}$	PiecewiseT	IndepNotAscending, SeqSizeMismatch
minD		$\mathbb{R}$	
maxD		$\mathbb{R}$	
x_1	–	$\mathbb{R}$	
x_2	–	$\mathbb{R}$	
feval	$x:\mathbb{R}$	$\mathbb{R}$	OutOfDomain

Semantics

State Variables The state variables are described below.

x_1: $\mathbb{R}$ # *first breakpoint*
x_2: $\mathbb{R}$ # *second breakpoint*
f: $\mathbb{R} \rightarrow \mathbb{R}$ # *piecewise polynomial*
minx: $\mathbb{R}$
maxx: $\mathbb{R}$

State Invariant None

Assumptions None

Access Routine Semantics new PiecewiseT($x:\mathbb{R}^n$, $y:\mathbb{R}^n$, $x_1^{\text{init}}:\mathbb{R}$, $x_2^{\text{init}}:\mathbb{R}$):

- transition: # *changing the values of x_1, x_2 and f according to the following steps using local real variables a_1, b_1, c_1, d_1, a_2, b_2, c_2, a_3, b_3, c_3, d_3, e_3, f_3*

  1. Additional local variables for independent variable: $x_0 = x[0]$, $x_1 = x_1^{\text{init}}$, $x_2 = x_2^{\text{init}}$, $n = \text{size}(x)$, $x_4 = x[n-1]$, $x_3 = x[\text{indexOf}((x_4 - x_2)/2)]$ # *x_3 is midway between x_2 and x_4*
  2. minx = x_0
  3. maxx = x_4
  4. Additional local variables for dependent variable: $y_0 = y[0]$, $y_1 = y[\text{indexOf}(x_1, x)]$, $y_2 = y[\text{indexOf}(x_2, x)]$, $y_3 = y[\text{indexOf}(x_3, x)]$, $y_4 = y[n-1]$
  5. Initial guess for parameters for first polynomial (assume linear): $a_1 = b_1 = 0; d_1 = y_0; c_1 = \frac{y_1 - y_0}{x_1}$

6. Initial guess for parameters for second polynomial (assume linear): $a_2 = b_2 = 0; d_2 = y_1; c_2 = \frac{y_2-y_1}{x_2-x_1}$
7. Initial guess for parameters for third polynomial (assume quadratic): $a_3 = b_3 = c_3 = d_3 = 0;$
   $e_3 = \frac{(y_3-y_2)(x_4-x_2)-(y_4-y_2)(x_3-x_2)}{(x_3-x_2)^2(x_4-x_2)-(x_4-x_2)^2(x_3-x_2)}$
   $f_3 = \frac{(y_3-y_2)^2(y_4-y_2)-(x_4-x_2)^2(y_3-y_2)}{(x_3-x_2)^2(x_4-x_2)-(x_4-x_2)^2(x_3-x_2)}$
   $g_3 = y_2$
8. $p^{\text{init}} = [x_1, x_2, a_1, b_1, c_1, d_1, a_2, b_2, c_2, a_3, b_3, c_3, d_3, e_3, f_3]$
9. p = optimize.curve_fit(ftrial, x, y, p^{init})
   Where,
   - ftrial is the function which has the polynomials which represent the three sections.
   - x is the time array
   - y is the temperature array
   - p^{init} is the initial guess values for the coefficients
   - This function returns the optimized list coefficients for the polynomials in ftrial function.

10. $x_1 = p[0]$
11. $x_2 = p[1]$
12. $f = \lambda x$: ftrial(x, p)

- output: $out :=$ self
- exception: $(\neg\text{isAscending}(x) \Rightarrow \text{IndepNotAscending} || |x| \neq |y| \Rightarrow \text{SeqSizeMismatch})$

minD():

- output: $out :=$ minx
- exception: None

maxD():

- output: $out :=$ maxx
- exception: None

x_1:

- output: $out := x_1$

- exception: none

x_2:

- output: $out := x_2$
- exception: none

feval:

- output: $out := f(x)$
- exception: $(\neg(\text{minx} \leq x \leq \text{maxx}) \Rightarrow \text{OutOfDomain}))$

Local Functions ftrial(x, x_1, x_2, a_1, b_1, c_1, d_1, a_2, b_2, c_2, a_3, b_3, c_3, d_3, e_3, f_3)
$p_1 = \lambda x : a_1x^3 + b_1x^2 + c_1x + d_1$
$d_2 = p_1(x_1)$
$p_2 = \lambda x : a_2x^3 + b_2x^2 + c_2x + d_2$
$g_3 = p_2(x_2)$
$p_2 = \lambda x : a_3x^6 + b_3x^5 + c_3x^4 + d_3x^3 + e_3x^2 + f_3x + g_3$
return $((x < x_1) \Rightarrow p_1(x)|(x_1 \leq x < x_2) \Rightarrow p_2(x)|(x \geq x_2) \Rightarrow p_3(x))$

indexOf($x^* : \mathbb{R}$, $x : \mathbb{R}^n$)
indexOf(x^*, x) $\equiv i$ such that $x[i] \leq x^* \leq x[i+1]$ # *maybe select closest x instead?*

Considerations The fitting for the piecewise curve depends on determining a good initial guess for the parameters. Section 3.2.5.1 provides an overview of how the initial guess can be determined.

The MIS of the Piecewise module consists of the following important fields - Name of the module, Uses, Syntax, Semantics, Local functions and considerations. The Name and Uses section are general information that mention the name of this module and the modules used by Piecewise module respectively. The Syntax represents the type of data structure of the output which is a user-defined data type in this case and the list of exported access programs. The semantics section includes the state variables and the assumptions. The structure of the access routine are also described in detail including transitions if any, output and exceptions.

3.2.5.2 Temperature module

The primary purpose of this module is to find the temperature and its derivatives across the cylinder at any time t and at all locations y. It uses the Piecewise module to find the temperature at any time t at all thermocouple locations and then uses curve fitting to compute the temperature and its derivatives at the location of interest. In this module, we had different potential options for curve fitting, such as regression, spline, interpolation etc. Our design for Temperature module had to accomodate these changes.

We started with regression and tried to fit one polynomial to the list of locations and temperature at those locations at any time t. However, this approach failed because of two reasons:

1. The temperature values obtained failed during visual inspection, because the temperature at all time t, for every non-thermocouple location, say $T_{0.5}$, which is between the thermocouple locations T_0 and T_1, did not always lie between the temperature values of the thermocouples. The correct values for $T_{0.5}$ are shown in Figure 3.7.
2. At higher locations (greater location values), the polynomial fit had oscillations that further decreased the quality of the fit.

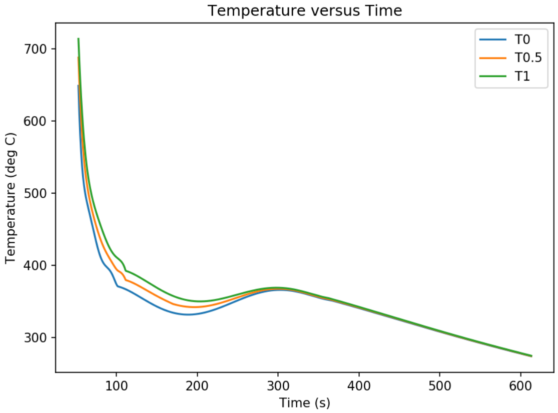


Figure 3.7: Expected temperature output for $T_{0.5}$ between T_0 and T_1

Due to these problems, we changed our curve fitting approach to "Interpolation" instead of "Regression". This improved the results of temperature prediction.

To find the derivatives of temperature at any time t and any location y across the cylinder, we used standard finite difference formulas from mathematics. These formulas could change if higher order differences are needed in the future to reduce the error. Therefore, the Temperature module was designed to accommodate these likely changes. The Module guide of the Temperature module is given below.

3.2.5.2.1 Module Guide of Temperature Module

Secrets: Algorithm for finding temperature values and their derivatives.

Services: Finding temperature values at any given location and time and finding derivative of temperature with respect to time and location.

Implemented By: SFS

In Temperature module, the secret is the curve fitting technique used to find the temperature at any given time t and at location y. This means that the curve fitting technique may change from interpolation to splines, or back to regression, but the module will continue to provide the temperature prediction and its derivatives irrespective of any change to the algorithm encapsulated in the secret. This is further explained with the "Module Interface Specification" shown below.

3.2.5.2.2 MIS of Temperature Module

Module TData

Uses config, PiecewiseADT for PiecewiseT, Load, SeqServices

Exported Constants $\Delta t = 1 \times 10^{-5}$
$\Delta y = 1 \times 10^{-3}$

Syntax

Exported Access Programs

Name	**In**	**Out**	**Exceptions**
TData_init			
TData_add	s : PiecewiseT, y : $\mathbb{R}$, t_L^* : $\mathbb{R}$, T_L^* : $\mathbb{R}$, $t_S^* : \mathbb{R}$, $T_S^* : \mathbb{R}$		IndepNotAscending
TData_rm	$i : \mathbb{N}$		InvalidIndex
TData_getC	$i : \mathbb{N}$	PiecewiseT	InvalidIndex
TData_getLiq	$i : \mathbb{N}$	$\mathbb{R}$, $\mathbb{R}$	InvalidIndex
TData_getSol	$i : \mathbb{N}$	$\mathbb{R}$, $\mathbb{R}$	InvalidIndex
TData_slice	$t : \mathbb{R}$	seq of $\mathbb{R}$, seq of $\mathbb{R}$	
TData_breakPts		seq of $\mathbb{R}$, seq of $\mathbb{R}$, seq of $\mathbb{R}$	
TData_T	$y : \mathbb{R}$, $t : \mathbb{R}$	$\mathbb{R}$	OutOfDomain
TData_dTdt	$y : \mathbb{R}$, $t : \mathbb{R}$	$\mathbb{R}$	OutOfDomain
TData_d2Tdy2	$y : \mathbb{R}$, $t : \mathbb{R}$	$\mathbb{R}$	OutOfDomain

Semantics

State Variables
S: sequence of PiecewiseT
Y: sequence of $\mathbb{R}$
t_L: sequence of $\mathbb{R}$
T_L: sequence of $\mathbb{R}$
t_S: sequence of $\mathbb{R}$
T_S: sequence of $\mathbb{R}$
Where,
- S is the sequence of PiecewiseT, which is a data structure storing information about a thermocouple such as the coefficients of the polynomials, break points for the sections etc. etc.
- Y is the locations at which the thermocouples are placed.
- t_L is the time at which the thermocouples reach the liquidus point.
- T_L is the temperature at which the thermocouples reach the liquidus point.
- t_S is the time at which the thermocouples reach the solidus point.
- T_S is the temperature at which the thermocouples reach the solidus point.

State Invariants None

Assumptions TData_init() will be called before any other access programs.

Access Routine Semantics

TData_init():

- transition: $S, Y, t_L, T_L, t_S, T_S := <>, <>, <>, <>$
- exception: none

TData_add($s, y, t_L^*, T_L^*, t_S^*, T_S^*$):

- transition: $S, Y, t_L, T_L, t_S, T_S := S || < s >, Y || < y >, t_L || < t_L^* >, T_L || < T_L^* >, t_S || < t_S^* >, T_S || < T_S^* >$
- exception: $exc := (|Y| > 0 \wedge y \leq Y_{|Y|-1} \Rightarrow \text{IndepNotAscending})$

TData_rm(i):

- transition: $s := s[0..i-1] || s[i+1..|s|-1]$
- exception: $exc := (\neg(0 \leq i < |S| - 1) \Rightarrow \text{InvalidIndex})$

TData_getC(i):

- output: $out := S[i]$
- exception: $exc := (\neg(0 \leq i < |S| - 1) \Rightarrow \text{InvalidIndex})$

TData_Liq(i):

- output: $out := t_L[i], T_L[i]$
- exception: $exc := (\neg(0 \leq i < |S| - 1) \Rightarrow \text{InvalidIndex})$

TData_Sol(i):

- output: $out := t_S[i], T_S[i]$
- exception: $exc := (\neg(0 \leq i < |S| - 1) \Rightarrow \text{InvalidIndex})$

TData_slice(t):

- output: $out := \langle Y_0, Y_1, ..., Y_{|Y|-1} \rangle, \langle S_0.\text{feval}(t), S_1.\text{feval}(t), ..., S_{|S|-1}.\text{feval}(t) \rangle$

- exception: None

TData_breakPts():

- output: $out := \langle Y_0, Y_1, ..., Y_{|Y|-1} \rangle, \langle S_0.x_1, S_1.x_1, ..., S_{|S|-1}.x_1 \rangle, \langle S_0.x_2, S_1.x_2, ..., S_{|S|-1}.x_2 \rangle$
- exception: None

TData_T(t, y):

- output: Find *out* using the following steps:
    1. Using TData_slice(), find the temperature at time t across all the thermocouples.
    2. Interpolate the temperature data from TData_slice() and the location values to find an interpolating polynomial of degree three (3).
    3. Evaluate the interpolant at location y to find the required *out* value.
- exception: $exc := (\neg \text{isInBounds}(Y, y) \Rightarrow \text{OutOfDomain})$

dTdt(t, y):

- output: $out := \frac{T(y,t+\Delta t) - T(y,t)}{\Delta t}$
- exception: $exc := (\neg \text{isInBounds}(Y, y) \Rightarrow \text{OutOfDomain})$

d2Tdy2(t, y):

- output: $out := \frac{T(y-\Delta y,t)+T(y+\Delta y,t)-2*T(y,t)}{\Delta y^2}$
- exception: $exc := (\neg \text{isInBounds}(Y, y) \Rightarrow \text{OutOfDomain})$

3.3 Regression Testing

Regression testing is the process of re-running the functional and non functional test suite after a change to the software to make sure that the previously developed and tested software still performs after a change. In our case study, we employed regression testing as we had to make several changes to SFS during its development. Some of the changes were due to repeated trial and error for optimization of experimental parameters. Every time, a change was made, we used pytest to run the entire test suite to ensure that the change did not break any code that was previously working. This technique also allows us to specify the permitted uncertainty between the predicted and actual data. Different forms of errors such as absolute error and relative error can be used to quantify the uncertainty.

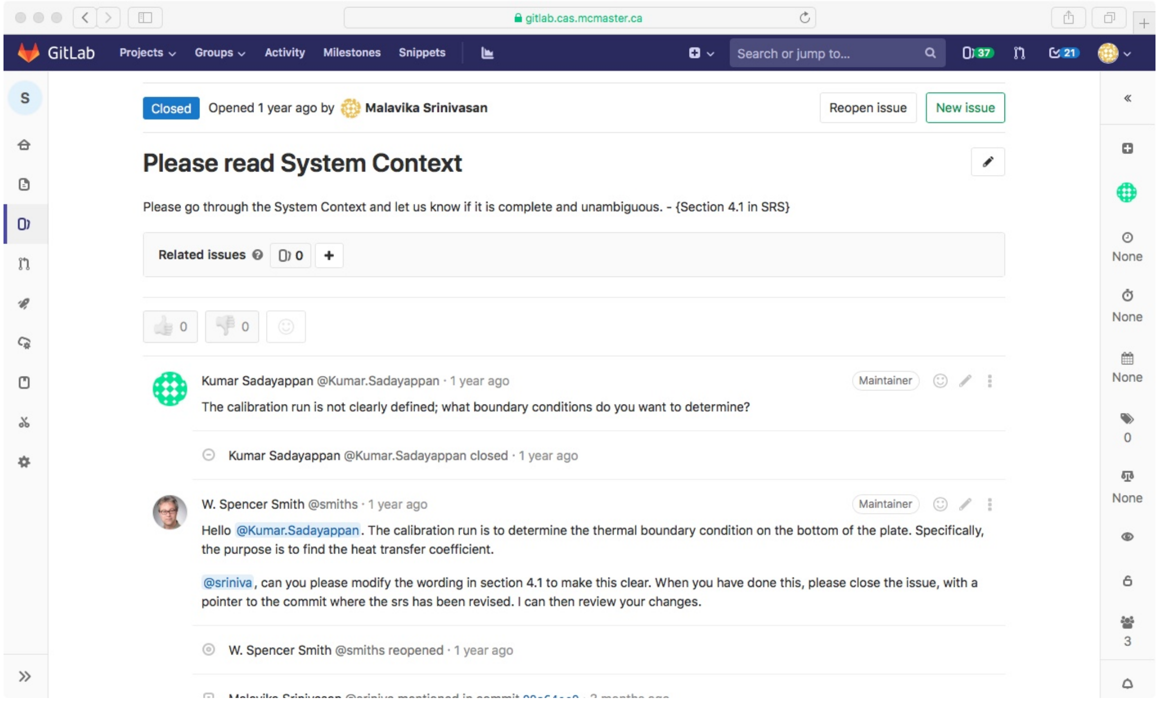
3.4 Task based inspection

“Task-based inspection” is a technique used by Kelly and Shepard [39]. We adopted this approach. We initially started with sending out the documents for review during our project meetings, but were unable to obtain any feedback. It was at that point we started our search for an efficient inspection technique and came across “Task-based inspection.” We were excited about this approach as it requires focused effort from the scientist, which can mean a lower time commitment. We decided to assign a task list to the scientist matching their skillset, as suggested by Kelly and Shepard [40]. The complexity of the task was kept simple so as to require the least amount of reading. The task list included questions such as “Please review the system context and let us know if there is any ambiguity.” The feedback to this task would require them to read the system context and let us know if they found any ambiguity, or anything else noteworthy. The list of questions developed for the inspection of the SRS of SFS is available in the appendix (C). The primary aim of employing this approach was to initiate communication and get them started in document review. This approach was successful, and we could see the scientist participating in document review. In Figure 3.8, we present a screenshot of the communication between us and the scientist in the issue tracker.

3.5 Git and Issue Tracker

Git is a version control system that is used for tracking changes in a computer files and coordinating work on those files among multiple people. It is primarily used for source-code management in software development. In our case study, we had to experiment with different numerical approaches using trial and error method and compare the results for optimization. Hence we used Git for branching and version control. At one point during the project, capstone students developed the GUI for SFS and helped with transitioning the code to Python 3.5, since it was originally written in Python 2.7. At that point, we had five developers in total, and Gitlab provided us with the necessary infrastructure to work independently on the shared code.

The Issue Tracker is a feature in Gitlab which is used mainly for communication and task management between people involved in software development. It offers a great platform for assigning and reporting bugs and sharing documents related to a specific task. In our case study, we successfully used the issue tracker for several purposes, such as workflow management between project members, conducting document review (see section 3.4), and communicating between team members. The issue tracker allowed us to assign issues to a person, notify them and create a deadline for



Closed Opened 1 year ago by Malavika Srinivasan

Please read System Context

Please go through the System Context and let us know if it is complete and unambiguous. - {Section 4.1 in SRS}

Related issues 0

0 0

Kumar Sadayappan @Kumar.Sadayappan · 1 year ago

The calibration run is not clearly defined; what boundary conditions do you want to determine?

Kumar Sadayappan @Kumar.Sadayappan closed · 1 year ago

W. Spencer Smith @smiths · 1 year ago

Hello @Kumar.Sadayappan. The calibration run is to determine the thermal boundary condition on the bottom of the plate. Specifically, the purpose is to find the heat transfer coefficient.

@sriniva, can you please modify the wording in section 4.1 to make this clear. When you have done this, please close the issue, with a pointer to the commit where the srs has been revised. I can then review your changes.

W. Spencer Smith @smiths reopened · 1 year ago

Malavika Srinivasan @sriniva mentioned in commit [illegible] · 2 months ago

Figure 3.8: Task based inspection

a task, thereby improving accountability. This was greatly helpful during document review.

Chapter 4

Learning and Observations

In this chapter, we present the experiences and learnings, from our perspective as software developers, in applying SE practices and principles to SC. We classify our experiences into four (4) major areas: attitude of the scientists, scientific computing, software engineering, and numerical techniques.

One of the observations, related to the myth that upfront requirements are infeasible for SC software, was significant enough that a complete chapter has been devoted to it. Further details about this observation can be found in chapter 5.

The content of the current chapter is organized as follows:

1. The observation is first presented.
2. In cases where our observation coincides with the observations in the literature, we summarize the literature to reinforce the observation and present our personal experience from this case study.
3. In cases, when our observations differ from literature, we present our personal observations, discuss what literature has to offer regarding this observation and explain potential reasons why they contradict the existing literature.

4.1 Attitude

Working on something in which the scientists are interested is not enough
In a previous study conducted [74], it was recommended that involving the code owners (scientists), from the beginning, before any requirements or code has been written, will facilitate a more complete understanding of the document driven development process and SE tools. The study suggests that the transfer of SE knowledge would be more feasible this way. This study is a successor to the previous study. So, we followed the recommendation and involved the scientists from the beginning.

Unfortunately, we observed that the scientists interest and our interest were conflicting with each other. We found that the scientists tend to be interested in reestablishing their scientific knowledge with the help of software. According to [35], the primary intention of software development in computational science is not to produce software but to obtain scientific results. They cling on to the working modules, but do not necessarily exhibit interest in developing high quality software, especially qualities such as maintainability, reliability, reproducibility and verifiability. This leads to a developer spending significant amounts of time implementing small changes. As discussed in chapter 3, the SE practice “Design for change” and proper documentation are the critical items that need to be prioritized for achieving the above mentioned qualities. However, the scientists tended to ignore the SE practices, leading to the development of software that was not maintainable. In some cases, it is possible that the entire software needs to be redeveloped when a small change is required.

This attitude of the scientists is partly due to the difficulty in explaining the consequences of not developing good quality software. The participants of the case study were all scientists and during the meetings they were more interested in discussing the scientific theory and the conversations inevitably went towards the equations and the numerical algorithms for implementing them. It was difficult to find time to talk about requirements, design and testing. All the scientists were certainly interested in the idea of maintainability, but with the demands on their time, and the need for results, their actions did not always match their interest. Other reasons for the lack of interest in software qualities can also be attributed to their lack of exposure to software qualities and skills. It was revealed in the post development interview, discussed in chapter 6, that some of our scientific partners had little knowledge of any SE development tools or practices.

Maintainability plays no role for scientific partners In developing a SC application, our scientific partners thought that their task ends with obtaining a working software. Hence, they were interested in writing the code, but showed less interest in proper software design and documentation. As an instance, initially we were working with a prototype of SFS, which was just a monolithic code without proper design. This was subjected to severe trial and error experiments requiring changes in the code. We spent considerable amount of time and energy making those changes as it was not designed for change. This could have been avoided if we spent some time designing the software before developing a prototype. However, scientists did not seem bothered about this and were driven by the demand for results. This is also pointed in [35], where they state “While domain knowledge is considered intellectual capital, software development knowledge is merely a technique, which consequently renders all technical decisions comparably unimportant . . . In their mind, source code is a more or less direct representation of the underlying scientific theory". A similar observation has been made in the article [26] in which the author says that "because the goal of a scientific software developer is the creation of new scientific knowledge, the emphasis placed on software qualities like correctness, reliability and maintainability are all very low". Hence, with more emphasis on implementation, and less on the quality aspect of the software, the software becomes unmaintainable.

In the early phase of this project, following the emphasis laid by the scientists on the code, we developed a prototype of the software which was subjected to trial and error experiments in fitting the experimental data. The results were not reliable because of a lack of confidence in the code, as no proper testing system was in place. Having proper design and documentation makes the maintenance of the software easier and reduces the overall risk of software failure. However, as long as the software produced scientific results close to the desired output, the scientific partners did not seem bothered about this.

This led to a series of experiments with no proper conclusion, as there was no way to measure the impact of a change. The testability and modifiability are important characteristics that make software maintainable, but when time pressures mounted, the noble goal of maintainability was one of the first casualties.

Scientists's hesitancy to learn SE practices The domain experts are not particularly interested in learning SE practices and implementing them. For instance, in our project we used Git, a tool for version control and tried to train the domain experts in using it. However, they did not try using Git, because from the perspective of the scientists, it was a low priority when competing against other demands for their time. A similar observation is mentioned in [42] that "Software Engineering skills are not appreciated by the scientists, instead viewed as an excessive demand to their already overwhelming job". In [82], the author has also mentioned the same problem: "Generally, scientists do not learn about software engineering techniques from software engineers". However, one exception to this is the "issue tracker". Scientists were enthusiastic about the issue tracker in Gitlab. This is discussed in section 4.3.

Verification and validation in scientific computing SC applications usually run simulations or compute output for which the correct solution is not yet known. This makes the verification and validation process difficult for an SC application. In [71], the authors mention that " Scientists often lack test oracles - real data against which they can compare their software's output". In the article [10], one of the employees in their case study, identified that inability to come up with good test cases makes the verification and validation process more difficult for scientific computing applications. In [80], it is mentioned that testing SC software is difficult because the correct solution is not known.

We believe that the attitude of the scientists towards testing needs changes. More creativity is need to test SC applications. For instance, when we were testing SFS, we found it hard to find a proper test case. We came up with tests for verifying the computation of gradients in one of our modules on our own, because we had trouble making testing a priority with the scientists. However, our "fake" experimental data was not that similar to the actual data because we did not include the discontinuity in the slope, because of the mathematical difficulty of differentiating the closed form solution over the discontinuity. It was not until toward the end of the project that the lead scientist mentioned the use of the error function to approximate the sudden changes of slope in the data. If the scientists had helped us with the test case a little earlier, we could have tested SFS comprehensively with a more realistic test case.

Alternatively, we can also identify the critical modules at the beginning of the project and discuss with the domain experts about ways to verify these modules. Testing methods such as metamorphic testing, which employs properties of the data or output, known as metamorphic relations, to generate test cases, can be employed to detect and eliminate errors. In the article [77], the authors suggest documenting a "solution validation strategy". Four potential evaluation strategies include:

1. solve the problem by different techniques, such as using an electronic spreadsheet, graphical solution etc;

2. substitute the calculated results back into the original governing equations to calculate the residual error;

3. partially validate the problem by validating simpler subsets of it for which the solution is known;

4. check that the governing equations are satisfied, boundary conditions are satisfied, energy is conserved, mass is conserved, etc.

Similarly in [48], the authors suggest a model for testing scientific software by considering only critical modules and leaving out non critical modules. It is obvious that along with the change in the attitude of the scientists towards testing, more effort and creativity is needed for verification and validation of SC software.

Science behind the software should also be verified When the software did not produce desirable results, the scientists questioned the numerical implementation. While the possibility of the numerical implementation being wrong should not be ignored, the possibility of underlying equations and other scientific inputs being wrong should also be considered while debugging. When SFS was developed and did not produce desirable results, the possibility of science being wrong or missing any details was not even considered. Even after repeated trial and error experiments with different numerical methods, the scientists did not think about the possibility of a flaw in the theory. In addition to the above two factors, the quality of the experimental data also plays a major role in the output from the SC software. After we tried changing the numerical techniques with no big change in the output, it was found that the quality of the input data also plays a major role in the output. For instance, the output from SFS had unexplainable anomalies with noisy data, which disappeared with better quality data.

The author's observation in the article [66] and [17] are exactly the opposite of what we experienced. In [66], it is mentioned that "All the scientists we interviewed doggedly pursued causes for their output not matching the oracle, but they focused on the theory, not the code". A similar observation was quoted in [35], where they say that the scientists judge model and algorithmic defects to be of far greater significance than coding defects.

We feel that the difference in our observation from the one in the literature may be due to three reasons. One of them being the scope of this case study. We had only a limited number of scientists in the team (seven) and not all of them were actively involved. So, our observation cannot be generalized. The other reason could be the confidence in the scientific model which eventually led the scientists to look for defects in implementation. The third reason could be the lack of awareness about the impact of quality of the input data in producing the output. This is discussed in detail in section 4.3. It turned out that the anomalies in input data were the reason for the unexpected output from SFS, which could have been avoided with more emphasis on metamorphic testing to access the quality of input data.

Scientists follow agile methodologies Agile methodologies are better accepted by scientific and engineering code developers than more traditional methodologies [10]. While working with the domain experts to automate the detection of phase change points, the algorithm developed by the scientist was designed to work only for the data set in consideration. The idea was to check if the same algorithm would work for other data sets too, if not then it could be modified until it works. This shows a typical agile fashion of developing a software. A similar observation was made by the authors in [17] in an ethnographic study conducted on the software development process at Hadley climate research centre, UK. It was noted that they used the agile software development approach combined with code reviews, version control management and extensive verification and validation strategies tailored specifically to their domain.

4.2 Scientific Computing

Common ground for communication has always been a problem Through out the life cycle of this project, we always had difficulties in interacting with the scientists. For instance, when we explain the functions of a module, we had to introduce SE terminologies such as user-defined data types, interface specification, state variables etc. But the domain experts had difficulties in understanding these terms. Similarly, scientists explained their requirements using scientific terminologies and we had difficulties in understanding those terms. This was also mentioned in [80] as one of the reasons why SE methods cannot be applied to scientific computing. In [35], the author mentions that "computational science and software engineering has established distinct terminologies. They need a common platform to understand each party clearly. Software engineers speak regarding software, requirements, qualities etc. whereas scientists talk concerning code."

An interdisciplinary team is essential During the development of SFS, we always felt the need for a partner in the development process, especially developers with domain knowledge; it could have resulted in better skill transfer. For example, when we were conducting trial and error experiments with the prototype of SFS, we had to change the code which required considerable amount of time and energy. We felt that the scientists were not bothered about this because they only analyzed the results from the software but did not change the code. If we had a domain scientist as a partner in coding, it would have been possible to emphasis the importance of design before implementation as they will experience the painstaking effort associated with changing the code. It was mentioned in [35] that the shift toward larger, interdisciplinary teams improves the often-ignored aspects of software such as modularity, maintainability, and team coordination. In the following articles [10, 72, 67], the authors emphasize that software developers and scientists can be partners in developing code.

Scientists tend to choose code language based on their familiarity Scientists decide the programming language based on their familiarity and not based on the requirements of the project [35]. Scientific code should be written in high level language for known benefits like better readability, abstraction, fewer lines of code etc., which in turn makes debugging easier [13]. When we decided that we wanted the code to be implemented in Python, the scientists were more inclined towards FORTRAN, as they were familiar with the latter programming language. We had to convince them to continue with Python. Our observations match those of [72]: "the scientist's choice for which language and environment used falls to familiarity".

Plots versus numbers Scientists are comfortable in analyzing the data using visual plots rather than a systematic data analysis procedure. In SFS, we determine the fraction solid concerning time and temperature from a temperature history data set. It was difficult for the scientists to determine the phase change points, or any other points of interest, when the data was presented as a file. However, when the results are presented as a plot, it does not need further processing to review the result, and it gives a glimpse of the entire data. Plots also have an advantage that you can find the major anomalies at the first glimpse without much processing. These factors make it suitable for scientific software development where frequent review and analysis of output data is common due to its exploratory nature. However, it should not be entirely relied upon for assessing correctness without further analysis. In [35], the authors mention that “Visualization of output data is the most common tool for verification and validation purposes”. However, they also caution against relying on it as a testing strategy and suggest to use it as a sanity check as mentioned in [8].

Conservative constraints, early testing It is essential to obtain conservative and correct constraints for testing. For instance, in the requirements specification document we have a section called physical constraints and software constraints on the inputs and calculated values. We have a wide range of allowed values for the parameters listed in the constraints section. When the typical value of the material properties are in the range of 1×10^{-8}, the acceptable range need not be as wide as 0 to 100. If we had stricter constraints such as 0 to 1, then it will act as a preliminary check for potentially incorrect calculations.
Similarly, a priori identification of some of the constraints or expected trends in the data plots would have helped with testing. Even though, the correct answer is not known, there are still properties of a correct solution that were known since the beginning. For instance, if we had discussed it earlier, we could have automatically tested for the intersection of temperature data from the thermocouples in the plot, which is a sign that the data is incorrect. This was a symptom that eventually led to diagnosing a mis-calibration in the data. If we paid more attention to the constraints, metamorphic testing could have helped us to identify the data anomalies missed by visual inspection.

4.3 Software Engineering

This section describes the lessons learnt during the development of software for scientific computing. These lessons are specific to software engineering methods and strategies.

Design log in issue tracker During the development of SC software, the practice of trial and error is inevitable. Therefore, we should design for change and follow a systematic approach. For instance, we had several meetings with the scientists during which various scientific and mathematical approaches were suggested to fit the experimental data. Even though we did trial and error experiments, we did not follow a systematic approach and hence it did not lead to a proper conclusion. Alternately, we could have derived better conclusions from the experiments by using the issue tracker and Git. The issue tracker could have been used as a trial and error log and Git branching could have been used to make the experiments repeatable. Each trial could be encapsulated in a issue and a Git branch following "separation of concerns" and grouped for similarity using features like "milestones" and "related issues". This prevents us from doing trial and error experiments in a spiral fashion. A similar idea is emphasized in the article [89], where the authors state that "It is essential to have a detailed record of the data manipulation and calculations".

Document review via likely changes During the development of SFS, we wanted the domain experts to review the software documents such as requirement specification, module guide etc. But, they showed less interest in document review than in the software prototype. It is not an uncommon practice in the software development to prefer prototypes to documentation. In the article [49], the authors have suggested that prototyping options make all life cycle models completely obsolete and even harmful. We suggest here an alternate idea since we cannot radically change the long standing attitude of the domain experts. We first need to identify the key modules of the project and start discussing the likely changes during the meetings, along with the working demonstration of the modules. This will get the scientists to participate in the development process as they get a chance to understand the design and working of the module. This will require them to understand the requirements of the software and will be an indirect way of reviewing the document, besides giving them an idea of modular programming, maintainability and software design.

This approach can be used to elicit information about a module as it gives them a chance to think about the input to a specific module, their types and likely changes. Frequent meetings combined with screen sharing applications and meeting software are essential for the success of this approach.

Scientists disregard command line Git While developing SFS, we used Git for version control. We tried to make the scientist use simple Git commands for pull, commit and push operations to run our code. Even after setting up the initial copy of the code and all the necessary modules in their machines, the domain experts were not interested to use Git through command line. Upon interacting with them, one of the expert users mentioned that “command line Git was hard to comprehend”. Git also has a web interface and we should have chosen the web interface to the command line interface to work with the scientists.

The same observation has been made by Greg Wilson [88] where he says “Git is a tool that researchers love to hate, with a vexing and confusing command-line interface intended more for seasoned programmers than casual”.

Scientists like the issue tracker During the course of SFS development, we used the issue tracker available in Git to manage the workflow and communicate with the domain experts [73]. Scientists liked the issue tracker and they said it was easy to use and helpful. A casual interaction with the scientist revealed that they felt it was better than email as they have all the information at one place for everyone. In the article [2], the authors also mention that “We needed an issue tracking system for formally recording bugs and communication between the teams”. In addition to the above mentioned points, the issue tracker also helped us to communicate with additional developers who worked in this project. The notifications, reminders and the ability to assign issues improved the accountability of the tasks assigned. It also helped us to create milestones and create issues under each milestone.

Task-based inspection While developing SFS, various documents were produced at different stage of development as part of the document driven approach. We wanted the domain experts to review the document but we obtained only limited feedback. Hence, as suggested in [39], we tried using a task-based inspection process to obtain feedback about the documents. We developed a list of questions, a sample of which is available as part of appendix C, for each scientist based on their area of expertise. Each question was put as a separate issue and was assigned to them using the issue tracker. The questions were framed in such a way that the scientists had to read a small section of the document and answer our question. The main aim was to get them started in reading the documentation. This approach proved successful. They answered our questions in the issue tracker and participated in the discussions about design decisions. Further interaction with them revealed that they liked this inspection, as it did not take much of their time and the questions were straight forward. The details about this method is given in section 3.4.

Design for Change In a SC application, changes are inevitable due to its exploratory nature. Hence, when designing a SC application, it is advisable to design for change. For instance, while developing SFS, the requirements document has been reasonably stable since the beginning because it was created in a abstract way, without implementation details. A SRS should say what to do but not how to do it [77].

All the design decisions were decoupled from the SRS which protected SRS against change when we wanted a change in the algorithm. As an example, when we tried different algorithms to slice through the cylinder to predict the temperature at any given time and location, our SRS remained valid because it just documented the need to find the temperature at any given time and location inside the cylinder and did not mention any details about the algorithm used. This is highlighted in the general definition below.

Number	GD4
Label	**Transformation of Experimental Data to Appropriate Function**
SI Units	–
Equation	$T(y,t) = \text{fit}(\mathit{Tdata}, y_{TC}, dt)$ where $\text{fit} : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^n \rightarrow \mathbb{R} \rightarrow (\mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R})$
Description	This general definition abstracts the concept of taking experimental data (Tdata) at the thermocouples over time and determining a function $T(y,t)$ that can be used in the instance models. In determining $T(y,t)$, it is assumed that the thermal resistance of the thermocouples can be ignored. The specifics of the transformation of the data to a function are left as part of the design of the numerical algorithm. The options include interpolation, regression, and using the data points directly. In the instance models, it is assumed that partial derivatives of $T(y,t)$ exist and can be calculated. When needed, these partial derivatives may be calculated directly from the experimental data, or by first finding $T(y,t)$ and applying mathematical operators to it. The symbols used in the equation for this general definition are defined as follows: $T(y,t)$ is a function that takes the position, as measured from the bottom of the cylinder, and the time and returns the temperature (°C). It represents the cooling curve over time t for all locations y. y is the distance from the bottom of the cylinder (m). t is the time from the start of data collection (second). fit() is a function that takes the thermocouple data Tdata, the locations of the thermocouples y_{TC} and the time step dt, and returns the appropriate function $T(y,t)$. Tdata is a 2D array of temperature readings. The columns correspond to each of the n different thermocouples, starting from the bottom and going up. The m rows correspond to the time of measurement. The start time for measurement is assumed to be 0 and the time between data points is assumed to be dt. y_{TC} is a 1D array of position values (in m) for the n thermocouples. dt is the time between experimental measurements of the time (second). m is the number of instants of time where the thermocouple data is measured. n is the number of thermocouples.
Source	–
Ref. By	DD6, DD7, IM1

Other documents such as the Module Guide (MG) and the Module Interface Specification (MIS) were also designed at the right level of abstraction. The MG lists only the modules, their secrets and services. It is abstract enough to hide the algorithm. For instance, we have a module called TData which predicts the temperature and its derivatives at a given time and location across the cylinder. We initially used a single polynomial to fit the temperature of each thermocouple but we were not satisfied with the results. Later we decided to use a piecewise polynomial to fit the data. These changes did not impact the requirements document and only the secrets of the module guide will undergo changes. This is the advantage of designing for change. The MIS contains all the details about the algorithm. When a new approach or a different algorithm is tried, the interface of the module remains stable, but it may be necessary to update the specification of the semantics.

Test driven development After developing SFS, we started designing the test cases. According to [10], "These issues combine to make the task of verification and validation for scientific and engineering applications very difficult. A member of the EAGLE team provided another reason why verification and validation is difficult: V&V is very hard because it is hard to come up with good test cases". Hence, we need to use creativity in designing the test cases. We can obtain information about the expected output through questions such as "Please draw the output you expect to obtain". This information can then be used in metamorphic testing to test SC application as most of them use plots as a preliminary check to test the software [35]. We could automate the test cases to check the metamorphic relations. For instance, in SFS, we could check that the temperature values at location 'y' lies between the data values at the thermocouple above and below. Metamorphic testing in combination with "Test driven development", where we begin writing test cases in parallel with the implementation, could be a better way to develop and test SC applications. In [73], the author propose to start software development by writing test cases even before the development process. This approach will help in improving the confidence in the code due to better testing approaches. Similarly, the authors of [2, 53, 62, 83] also advocate the test driven development for SC.

Faked rational design process In developing SFS, we followed a faked rational process design, as advocated in [80]. If we followed a pure waterfall model, we would have started by writing down the requirements upfront, which is a challenging task for a SC. After the requirement specification, we proceed to develop the MG and the MIS. At the end of each development phase, a document would have been produced which will need to be verified by the domain experts. But, the domain experts usually show less interest towards documentation as described previously. This would have resulted in delayed feedback or sometimes no feedback at all. If we proceeded with the implementation without document review, then the risk of software failure or software not being functionally suitable would be very high. Hence, this approach would not be suitable for SC applications where scientist place more value on code than the qualities of the software. This is also mentioned in the article [35] based on [17] that "Traditional software development processes that employ a big design up front approach such as the waterfall model are a poor fit for computational science".

As an alternative, faked rational process design was proposed [57]. In this approach of developing software, we can start with the implementation and present the documents as if the software was designed following a rational process. In a typical SC application, feedback and participation from the domain expert is critical for the success of the software. In the article [74], the author mentions that getting the scientists involved in the project from the beginning has positive potential. Without a prototype of the actual software, it is difficult to get the scientists involved in the development process. We followed a faked rational process design and document driven approach to develop SFS with the goal to involve the scientists since the beginning of the project. Hence, we started by implementing the critical modules of SFS. Even though we were not able to obtain active participation from the domain experts, they were quite accommodating when it came to interpreting the results and debugging the anomalies in the graph.

Scientists lack awareness about software quality and skills Scientist lack awareness about software quality. For an SC application, qualities such as correctness, confidence and maintainability are considered important. The scientist often perceives the code as a mere representation of the theory [35]. Hence, they do not concern themselves with the quality of the software. Sometimes, the domain experts think that if an algorithm works for a particular data, then it can be made to work for all the cases. But that is not always true. Without confidence in the code, correct results can sometimes be an illusion. For instance, when SFS was developed and the experimental data was converted to a function of temperature and time, we tried to automate the detection of phase change points. An algorithm was developed and implemented successfully for the data set in consideration. However, they did not work for other data sets. The above situation clearly portrays the lack of awareness about developing a reliable algorithm which will work for most data sets.

Similarly, scientists lack software skills. Their view about the software is limited to "code" without any thought about defined interface and reuse. In [13] the authors suggest that software skills must be taught during undergraduate or graduate courses so that scientist do not lack awareness about software tools, techniques and their benefits. In our case study, we observed the same through their attitudes and lack of interest when we introduced any software tools such as Git or in discussions about design aspect of the software such as “separation of concerns”, “design for change” etc. As an example, when SFS did not produce expected outputs, the domain experts helped us to analyze the wrong outputs. They preferred looking at a monolithic code to understand the design to reading the Module Guide and Module Interface Specification.

Mythical man effect Scientists prefer obtaining knowledge from coworkers to reading documents. This authors of [35] also mention that the scientists prefer informal, collegial ways of knowledge transfer to understand a piece of software than relying on its documentation. They find it harder to read and understand documentation artifacts than to contact the author of a particular part of the software and discuss their questions with them. As an example, when the first prototype of the software was developed, the domain experts wanted a copy of the code and preferred informal and casual meetings to ask questions about the code. They did not bother obtaining information about the software from the requirement document and module guide even after repeated emphasis and instructions from our end.

4.4 Numerical Methods

In this section, we present our observations concerning the application of numerical methods during the development of SFS.

Systematic trial and error Trial and error experiments are inevitable while developing a scientific computing software. However, it should be integrated with testing to have a meaningful outcome. As an example, we tried different algorithms for slicing through the height of the cylinder to find the temperature at a given location and time. All these approaches gave us outputs which differed only slightly from each other. We should have tested these approaches with the idea of quantifying the impact of using a certain approach. If we code anticipating changes, we can determine which algorithm is best suited and save time by only completing numerical experiments once. Further, these trial and error experiments can be incorporated into strategy design pattern to make it easier to switch between algorithms.

Difficulty in handling scientific data. SC significantly differs from computing in other domains in terms of their source of input data. In SC, we handle experimental data which is prone to uncertainties, such as human error, equipment malfunction etc. Correct input is crucial for a SC software. In the article [82], the author also mentions that “Many of the issues raised in the literature regarding software quality have also been identified in the wider context of data quality. The growth in the size of research data sets and software processing capabilities have led several researchers to consider quality from a data rather than software process perspective”.

As an instance, we found that SFS behaved better with the data obtained from industrial trials than the data from experiments in the lab. The quality of the data improved drastically with the high precision thermocouples, which in turn had a significant impact on the results from SFS.

Assessing input data In SC, we handle experimental data which is prone to uncertainties such as human error, equipment malfunction etc. A mere visual inspection may not be enough to ascertain the quality of the input data. Correct input is crucial for SC software. In the article [82], the author also mentions that “Many of the issues raised in the literature regarding software quality have also been identified in the wider context of data quality”. So, it is necessary to access the quality of the data.

As an instance, when we did not obtain expected output from SFS, we eventually determined that the source of the inaccurate results was actually the input data. To demonstrate this, we present the equation which is used to calculate the fraction solid.

$$\dot{f}_s(f_s,t) = \frac{C_v(f_s)}{L\rho(f_s)}\left[\frac{\partial T(t)}{\partial t} - \alpha(f_s)\frac{\partial^2 T(t)}{\partial y^2}\right]$$

In the above equation, it can be clearly seen that the calculation depends on the partial derivatives $\frac{\partial T}{\partial t}$ and $\frac{\partial^2 T}{\partial y^2}$. Hence, when the results were not correct, we decided to check the computation of $\frac{\partial T}{\partial t}$ and $\frac{\partial^2 T}{\partial y^2}$. We found that the second derivative $\frac{\partial^2 T}{\partial y^2}$ was not metamorphically correct based on the feedback from the scientist . The plots were supposed to be smooth and converge into x-axis at values closer to zero after solidus time. We traced the cause to be three reasons.

- The value of Δy in the formula for $\frac{\partial^2 T}{\partial y^2}$ needs to be greater than the χ^2 of the fit [87].
- The formula for $\frac{\partial^2 T}{\partial y^2}$ was changed. We were using a central difference formula, which was changed to a forward difference formula based on the suggestion from the scientist. This improved the results because previously the formula was using thermocouples which were in different phases such as 2 phase zone and solidus zone. With the new formula, the calculations were restricted to thermocouples which were in the same zone.

- The primary cause was however traced to be the actual data itself because the data from the thermocouples had anomalies which could not be identified with a brief visual inspection.

For instance a closer look at the data revealed that after 60 mm from the bottom, the data had anomalies which was the reason for the wrong metamorphic form of the $\frac{\partial^2 T}{\partial y^2}$ at locations greater than 25 mm. Once we identified this error, we were able to restrict our calculations until 60 mm and were able to produce expected output. This clearly indicates that the quality of the input data has a major impact on the outputs.

In this chapter, we have described various factors that can potentially impact the development and quality of SC software. Even though the lessons and experiences described were pertaining to the development of SFS, most of them are supported by literature references as well. This collection of experiences may be used by future developers as a reference during their experiments in applying SE practices into SC.

Chapter 5

Myth Busted

In the previous chapter, we discussed some of the experiences during software development for SFS. We found that one of the observations from the literature regarding upfront requirements specification can be characterized as a myth. In this chapter, we discuss the evidence for why we consider it as a myth.

The content of this chapter is organized into four (4) sections. In the first section, we present the observations from the literature which states that 'upfront requirements are not possible for SC'. In the second section, we present our experience in specifying requirements for SFS and present instances from the requirements document to explain the process. In the third section, we discuss the idea behind the design of SFS. In the last section, we demonstrate the approach we followed to identify the requirements at the beginning of the project.

5.1 Literature on Upfront Requirement Specification

Myth: "Upfront requirements are not possible for Scientific Computing"

It is a common observation in literature that "upfront requirements are not possible for SC". It is also often claimed that the above statement is true, not just for SC, but for any programming domain [49]. In this section, we will discuss some of the references in the literature that claims that upfront requirements are not possible for SC.

There are multiple references in the literature regarding the above myth. To quote a few:

- In article [35], it is mentioned that "In science, software is used to make novel discoveries and to further our understanding of the world. Since scientific software is deeply embedded into an exploratory process, you never know where its development might take you. Thus, it is hard to specify the requirements for this kind of software up front as demanded by traditional software processes".

- Segal and Morris [71] say that "Full up-front requirement specifications are impossible: requirements emerge as the software and the concomitant understanding of the domain progress".

- In the article [10] the authors state "While most scientific and engineering projects are ultimately based on the underlying laws of nature, which are fixed, the application of those laws to a specific problem is often unknown at the start of the project. Most requirements, beyond some obvious high-level ones, are discovered during the course of the project".

- In the article [67], Segal says that "The research scientists are experienced in developing their own software in the laboratory in a highly iterative manner, and having requirements emerge in succeeding iterations. They do not appreciate the need to articulate requirements fully and upfront as demanded by a staged methodology, and found this articulation very difficult to do".

- A similar statement was also made by the authors of [17] where they claim that "computational scientists generally adopt an agile development approach because they generally do not know the requirements up front".

- Segal mentioned that "Supplying requirements upfront ran counter to the previous experience of developing their own software in the laboratory" in a case study where software developers developed the SC software for the scientists [68]. Similarly in the article [66], the authors referenced a statement mentioned by one of their interviewees which says "None of our interviewees created an upfront formal requirements specification. If regulations in their field mandated a requirements document, they wrote it when the software was almost complete".

- In [69], the author articulates that "Requirements emerge, as the understanding of both the software and the science evolves" which emphasizes that upfront requirements are not possible.

5.2 Requirement Specification for SFS

In this section, we present our experience in identifying requirements for SFS and present our reasons to why we think the above observation discussed in section 5.1 is a myth. In [77], the authors presented a template for SRS, which was successfully used for different SC applications [80, 75, 74, 78, 76]. This template allows you to create the requirements document at proper level of abstraction which makes requirements specification easier.

For instance, in our project, we used this template and identified the critical requirements of the project and documented it in the requirements document. We did not mention any design decisions such as algorithms or specific numerical techniques which were to be used. The requirements of SFS from the SRS document is summarized below.

R1: Input the configuration and specification parameters.

R2: Input the temperature data collected from the thermocouples: *Tdata*, n, necessary material properties for the known alloy and the temperature of the environment: $k_S(T)$, $C_p^S(T)$, $\rho_S(T)$ and T_{env}

R3: Using the input information (R1 and R2) calculate the heat transfer coefficient $(h(t))$.

R4: After running the experiment, input the temperature data collected from the thermocouples: *Tdata*.

R5: Compute α_b and α_e.

R6: Input the additional information necessary to solve for f_s: L

R7: Using the temperature data (R4) and other information (R5 and R6) and configuration information (R1), calculate the value of $f_s(t)$.

R8: Using $f_s(t)$ and $T(t)$ find $f_s(T)$.

These requirements identified at the beginning have remained stable throughout the development process because we did not mention any details about how these requirements will be implemented. Similarly, in the section "General definition" in SRS, we stated that we want to translate the experimental data to a fitting function, but we did not specify the details of how this will be accomplished. This is highlighted in the general definition below.

This figure is repeated here for reader's convenience.

Number	GD4
Label	**Transformation of Experimental Data to Appropriate Function**
SI Units	–
Equation	$T(y,t) = \text{fit}(Tdata, y_{TC}, dt)$ where $\text{fit} : \mathbb{R}^{m\times n} \rightarrow \mathbb{R}^n \rightarrow \mathbb{R} \rightarrow (\mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R})$
Description	This general definition abstracts the concept of taking experimental data ($Tdata$) at the thermocouples over time and determining a function $T(y,t)$ that can be used in the instance models. In determining $T(y,t)$, it is assumed that the thermal resistance of the thermocouples can be ignored. The specifics of the transformation of the data to a function are left as part of the design of the numerical algorithm. The options include interpolation, regression, and using the data points directly. The symbols used in the equation for this general definition are defined as follows: $T(y,t)$ is a function that takes the position, as measured from the bottom of the cylinder, and the time and returns the temperature (°C). It represents the cooling curve over time t for all locations y. y is the distance from the bottom of the cylinder (m). t is the time from the start of data collection (second). fit() is a function that takes the thermocouple data $Tdata$, the locations of the thermocouples y_{TC} and the time step dt, and returns the appropriate function $T(y,t)$. $\vdots$ m is the number of instants of time where the thermocouple data is measured. n is the number of thermocouples.
Source	–
Ref. By	DD6, DD7, IM1

This general definition is further refined into an "Instance model" in the SRS where we discuss about GD4 from which the partial derivatives are calculated. The instance model is presented below which is also designed at proper level of abstraction.

Number	IM4
Label	**Solve Inverse Heat Transfer Problem for Heat Transfer Coefficient**
Input	$T(y,t)$ (see 5.2), from which $\frac{\partial T}{\partial t}$ and $\frac{\partial^2 T}{\partial y^2}$ can be derived, as required. Material property $k_S(T)$, $C_p^S(T)$, $\rho_S(T)$ T_{env}
Output	$h(t)$ such that the following PDE and boundary conditions are satisfied using the experimental data represented in $T(y,t)$: $$\frac{\partial T}{\partial t} = \alpha_S(T)\frac{\partial^2 T}{\partial y^2}, \text{ where } \alpha_S(T) = \frac{k_S(T)}{\rho_S(T)C_p^S(T)} \quad (5.1)$$ subject to the boundary conditions $q = 0$ on all boundaries, except for the bottom of the cylinder where $q(t) = h(t)(T_0(t) - T_{\text{env}}(t))$.
Description	The symbols used in this model are as follows: $T(y,t)$ is the temperature in °C found using the temperature values at the known thermocouple locations in (m) at time t (s) (from 5.2) $\vdots$ $\rho_S(T)$ is the density of the solid metal, potentially as a function of temperature ($\text{kg}\,\text{m}^{-3}$)
Sources	Some related information is available at: `http://web.cecs.pdx.edu/~gerry/class/ME448/notes/pdf/`
Ref. By	

The highlighted text in the instance model above shows that the $\frac{\partial T}{\partial t}$ and $\frac{\partial^2 T}{\partial y^2}$ needs to be calculated. However, it is not mentioned how the calculation will be done. There are different formulas that we can use to compute the partial derivatives.

In the sections from the SRS document presented above, we only describe what to do but not how to do, as mentioned in [77]. For a typical SC application trial and error is inevitable due to its exploratory nature. Hence we need to design for change, which means to we should have the freedom to explore different numerical methods. Our SRS template is primarily focused on writing the requirements at proper level of abstraction.

Specifying upfront requirements for a SC application, even though feasible, may be a challenging task. However, with "Faked rational design process", it is possible to update the SRS document as it evolves, faking the documentation to look like we had a perfect understanding from the beginning. The benefits of this approach are discussed in section 4.3.

5.3 Design for Change in the MIS for SFS

In this section, we discuss the design for SFS and the software artifacts produced during the design phase. As an example, the MIS of the Temperature module is presented below.

In the MIS, the method T(y, t) gives the temperature at any time *t* and location *y*. The syntax of this method, given in section 5.3 does not change with changes in the algorithm, making the interface stable.

Similarly, the methods such as dTdt(t, y) and d2Tdy2(t, y) gives the derivatives of the temperature with respect to time and location. The formulas for computing the gradients may be changed but the interface of these methods does not undergo any change. We presented this document as if we knew the formula during the design phase but we actually faked it in this document by changing the formula once it was finalized.

Module Interface Specification of Temperature Module

Module TData

Uses config, PiecewiseADT for PiecewiseT, Load, SeqServices

Exported Constants $\Delta t = 1 \times 10^{-5}$
$\Delta y = 1 \times 10^{-3}$

Syntax

Exported Access Programs

Name	**In**	**Out**	**Exceptions**
TData_slice	$t:\mathbb{R}$	seq of $\mathbb{R}$, seq of $\mathbb{R}$	
⋮			
TData_T	$y:\mathbb{R}$, $t:\mathbb{R}$	$\mathbb{R}$	OutOfDomain
TData_dTdt	$y:\mathbb{R}$, $t:\mathbb{R}$	$\mathbb{R}$	OutOfDomain
TData_d2Tdy2	$y:\mathbb{R}$, $t:\mathbb{R}$	$\mathbb{R}$	OutOfDomain

Semantics

State Variables
S: sequence of PiecewiseT
Y: sequence of $\mathbb{R}$
⋮
Where,
- S is the sequence of PiecewiseT, which is a data structure storing information about a thermocouple such as the coefficients of the polynomials, break points for the sections etc.
- Y is the locations at which the thermocouples are placed.

State Invariants None

Access Routine Semantics
⋮
TData_slice(t):

- output: $out := \langle Y_0, Y_1, ..., Y_{|Y|-1}\rangle, \langle S_0.\text{feval}(t), S_1.\text{feval}(t), ..., S_{|S|-1}.\text{feval}(t)\rangle$
- exception: None

TData_T(t, y):

- output: Find *out* using the following steps:

  1. Using TData_slice(), find the temperature at time t across all the thermocouples.

2. Interpolate the temperature data from TData_slice() and the location values to find an interpolating polynomial of degree three (3).
3. Evaluate the interpolant at location y to find the required *out* value.

- exception: $exc := (\neg \text{isInBounds}(Y, y) \Rightarrow \text{OutOfDomain})$

dTdt(t, y):

- output: $out := \frac{T(y,t+\Delta t)-T(y,t)}{\Delta t}$
- exception: $exc := (\neg \text{isInBounds}(Y, y) \Rightarrow \text{OutOfDomain})$

d2Tdy2(t, y):

- output: $out := \frac{T(y-\Delta y,t)+T(y+\Delta y,t)-2*T(y,t)}{\Delta y^2}$
- exception: $exc := (\neg \text{isInBounds}(Y, y) \Rightarrow \text{OutOfDomain})$

5.4 Guide towards Requirement Specification

In this section, we present some of the approaches we followed in identifying the requirements for SFS. This can be viewed as a beginner's guide to identify the requirements for a SC application.

- We interacted with the scientists at the beginning not just to determine the requirements of SFS, but also to identify the likely changes. While we gathered information about the functional and non-functional requirements of SFS, we also focused on identifying and documenting the likely and unlikely changes. These discussions helped us to restrict the scope of the software and to design the software to adapt easily for the changes.
- We followed a faked document driven method to develop SFS. This has been discussed in detail in chapter 3 of this thesis. This approach gave us the freedom to add any necessary information in the requirement document at a later stage and fake the presentation of the document as if it was part of the original document.

However, it is worth noting that identifying the likely changes for a SC application may not be easy. We suggest the scientific developers identify the family of likely programs that their final program will be a part of, then it is feasible to document requirements, and to start this “up front”.

Chapter 6

Feedback on SE Tools and Techniques

In this chapter we present the feedback of the scientists on the SE principles and practices applied in developing SFS. The content of this chapter is organized into three sections namely methodology (6.1), scope (6.2), feedback (6.3) and summary (6.4).

6.1 Methodology

In this section, we describe the methodology for collecting the feedback. We conducted two (2) interviews:
- Pre-development interview
- Post-development interview

6.1.1 Pre-development Interview

In this section, we present the details about the pre-development interview. We interviewed the scientists at the start of the project and gathered information about their expertise, area of specialization, exposure to software skills and coding experience, if any. We discussed the list of qualities that are essential for SFS. In this process, we studied their attitude towards SE tools and techniques. All the scientists were interested in maintainability of SFS. Other qualities such as correctness and verifiability were also considered important for SFS. All of them had previous experience in working with scientific software. Some of them had developed code, but did not follow a software development approach. It was observed that some of them thought coding was the only activity in the software development process. The list of interview questions and the answers from one of the scientists is available in appendix (E).

6.1.2 Post-development Interview

In this section, we present the details about the post-development interview. We developed SFS using SE tools and principles mentioned in section 3.2. We involved the scientists from the beginning of the development cycle, even before any requirements or code was written, giving them an opportunity to witness the application of SE in developing SC software, as per the recommendation in [74]. To study their attitude towards SE tools and techniques, we decided to interview them and collect their feedback. The methodology was as follows.

- After the development of SFS, we presented the SE tools and techniques that we applied in developing SFS. This step was necessary because some of the principles like “Design for change”, were adopted during the design phase, considering the needs of the project. We did not involve the scientists in the design phase because during the pre-development interview (6.1.1), we realized that the scientists did not have prior exposure to software design and hence we were worried that scientists may find the task of designing the software arduous. Hence it was necessary to summarize all the principles and practices applied in developing SFS before asking for feedback.
- We had a set of questions which will be presented in section 6.3 and we requested the scientists to answer those questions.
- We analyzed the responses and feedback from the scientists to classify the SE approaches which were successful and to study the attitude of the scientists towards applying SE into SC.

6.2 Scope

In this section we discuss the scope of the post-development interview. This study is a successor to previous research carried out by Smith et al. in [74], in which existing SC software was redeveloped using SE approaches without involving the scientists during the development process. In this study, we planned to collect feedback from the scientists on applying SE approaches to SC by involving them during the development of SFS. Even though we had seven scientists in our project team, only two of them were actively involved during the development of SFS. Other members in the team of scientists contributed towards the scientific theory but their involvement towards the SE aspect of this case study was minimal.

We had a total of twelve (12) interviewees. The participants of the post-development interview are mostly academicians from a diverse background who were interested in scientific software development and were not just from the team of scientists in the case study. This implies that some of them were not a part of the project from the beginning. However, it was not necessary because the interview was aimed at collecting feedback on SE principles and practices applied in developing SFS, and not about the actual software. The value of this work would have been enhanced if we had additional interviewees from industry.

Some of the interviewees have been working with us since the beginning of this project and hence there is a possibility of an unconscious bias in their feedback. During the analysis of responses from the interviewees, we felt that some of the questions in the interview, even though unintentionally, were biased and may have led the scientists towards positive comments.

6.3 Feedback

In this section we present the feedback of the scientists on the SE principles and practices applied for developing SFS. The content of this section is organized into a list of questions posed to the scientists along with their responses and feedback obtained during the post development interview. The questions were primarily focused on studying their attitude towards applying SE into SC. The content of the presentation is available in the appendix (A).

Q1: What could go wrong when the software does not produce expected results?

1. Science
2. Code

A1: About 15% of the interviewees thought that 'Code' could be wrong and the remaining were inclined towards both options. It is worth noting that, the 15% of the interviewees who chose 'code' thought that by 'Science' we meant the basic proven scientific principles. However, by the word 'Science' we meant the equations and principles used to solve the scientific problem associated with the software. This was revealed when we asked for the reason behind their choice and upon clarification, they were also inclined towards the possibility of both potentially being wrong.

It is worth noting that this feedback is different from our observation presented in section 4.1. Our observation was based on working with only one (1) scientist and it cannot be generalized.

Q2: After this presentation, has your idea about software development changed from just writing code? Please explain?

A2: All of the interviewees answered 'Yes'. Upon further discussion, it was discovered that many of the scientists originally thought that software development was just writing code. One of our interviewees, who had previous experience in developing software mentioned that, 'I always felt the void while developing software and SE fills the void'. Another interviewee felt that learning SE helps them understand the rationale behind software decisions.

Q3: Were you aware of software skills and tools like design for change, separation of concerns, testing techniques, issue tracker, version control etc before the start of this project?

A3: About 75% of the interviewees said that they had previous experience in using version control. One of them had prior exposure with the Github workflow and doxygen comments. Some of the interviewees have had an exposure to tools such as issue tracker (bug tracker) and have used version control system. About 10% of our interviewees had absolutely no exposure to software skills. The only part of the software known to them was the code. .

Q4: Do you feel it would have been nice if there was an exposure about software skills during your graduate studies? If yes, how it would have been helpful.

A4: All of them unanimously agreed to this statement. One of our interviewees wrote that "Yes, nowadays, one cannot be successful without exposure to software". Another interviewee mentioned that she was taking a course on SE to get exposure to SE principles and practices. This feedback has reinforced the author's observation in [13].

Q5: If you want to incorporate any of the above mentioned steps in future software development practices, what would be it? Please choose from below:

1. SRS (Documenting science) (Let us assume you have automatic tools)
2. Module Guide (Module decomposition of code)
3. Issue tracker (Use issue tracker)
4. Testing

5. Design for change

A5: About 80% of the interviewees mentioned that they liked to implement all of the SE tools and practices which we presented. One of our interviewee mentioned that all the SE principles and practices suggested above are useful only if the SC software will be used by people other than the developer and the software will be used for a long period of time. Keeping in mind the effort required to apply the SE principles and practices such as document-driven development, it makes good sense to apply these practices to a software which will be used for a considerable amount of time.

It is also essential to note all the software documentation related to SFS was prepared by us and scientists had no idea how much of an effort it was to put them together. Their choice may have been different if we asked them to prepare the documentation.

In addition to this, it is worth noting that there is an inherent bias in our approach because the post-development interview was conducted with scientists who were either familiar with this project or known to us for a long time. It could have been avoided if we had more time and resources.

Q6: Do you think review of the documents would have increased your confidence in the software?

A6: Most of our interviewees appreciated document-driven design. Almost 90% of our interviewees mentioned that document review would have increased their confidence. One of them mentioned that "Yes, the review of all the documents together would have increased the confidence on the software". One of our interviewees expressed his concern over the test cases chosen. However, he was convinced when we mentioned that the test cases can also be verified from the Verification and Validation plan.

Q7: If you want to develop a scientific computing software, which scenario would be the best?

1. Scenario 1 : Scientists learning and using the software engineering tools and techniques? They can use existing templates for documentation and automatic documentation generator tools will also be available.
2. Scenario 2 : A software engineer understanding the scientific research (Please consider the wide range of scientific computing domain) and developing the software.

A7: About 80% of the interviewees mentioned that scenario 1 was a better choice. Generally, understanding the scientific research is considered as a daunting task by a software developer. In the article [10], the author mentions that "It will be easier for the scientists to learn code than for the developers to comprehend the science behind the project". However, some of our interviewees felt that it will be an additional burden on their work load. One of our interviewees mentioned that it should be a combined effort from both scientists and the software developers. Other notable feedbacks included having a multi disciplinary team and a combined effort by the team members would be the best. . This feedback has reinforced the observation in the following articles [35, 10, 72, 67] where they say that an inter-disciplinary team will be beneficial for developing SC software.

Q8: Can you make future changes in a software with just the code and no documentation?

A8: Almost 80% of our interviewees mentioned that it will not be possible to maintain software without documentation. Some of our interviewees mentioned that "It is possible to maintain the software even without any documentation, but it will be very difficult. However, future changes will be impossible".

Q9: With the given level of documents, do you think a new person could understand the software and continue the work?

A9: All the interviewees agreed that it was possible to maintain the software with the given level of documents. One of our interviewee mentioned that some training may be necessary in the program environment in addition to the documents.

Q10: Please share your feedback about issue tracker. Do you think it is better than email?

A10: All the interviewees unanimously mentioned that they liked the issue tracker. The reasons for the success of issue tracker as mentioned by the interviewees included easy record keeping and follow up. Some of them mentioned that it is easier to track information in an issue tracker when compared to email. However, one of our interviewees mentioned that issue tracker is suitable for small problems and smaller team size and not suitable for bigger problems running into pages with more number of developers working on the project. However, we believe that issue tracker is even more valuable for bigger softwares with multiple developers. Some of the advantages of issue tracker are discussed in chapter 2, which makes it explicit why it is suitable for a large scale project with several developers.

6.4 Summary

In this section we summarize the feedback of the scientists presented in detail in section 6.3. Almost all the interviewees who were scientists see the value of applying SE into SC. They are optimistic that SE will improve the quality of SC applications. However, when it comes to learning the SE approaches, we saw a decline in the number and they opted to have multi disciplinary team. Some of our interviewees mentioned that they liked the idea of applying SE approaches to SC but they were also worried about the learning curve of the SE practices and principles. Moving forward, it is clear that scientists see the value of SE approaches for SC but “lazy proof” practices and tools, that are easy to learn and use, without requiring much effort, are necessary for scientists to use SE in developing SC applications.

Chapter 7

Conclusions

In this chapter, we provide a summary of the thesis with concluding remarks and a list of potential future work.

7.1 Thesis Summary

This thesis has provided insight into applying SE practices to develop SC applications. The primary objective of this thesis is to improve the quality of SC applications by applying SE principles and practices. Often, the developers of the SC applications are the scientists themselves, who are also the end users of the software. Typically the professional end user developers have little or no education or training in SE [71]. Hence, it becomes necessary to motivate the end user developers, also referred as 'scientists', to use the principles and practices of SE to produce high quality SC software.

In a previous attempt to bridge the gap between SE and SC developers in [74], it was observed that the end user developers see the value of document driven design, but were not convinced with the amount of effort required. It was recommended to involve the scientific partners, from the beginning, before any requirements or code are written. This will facilitate a more complete understanding of the document driven development process and SE tools. Also, they [74] suggested that the transfer of SE knowledge would be more feasible this way. This study is a successor of the previous study. We involved the scientists in the software development process, giving them an opportunity to witness the application of SE practices and principles to develop an SC application. The methodology for conducting this case study is presented in section 1.2. This summary is organized based on the steps mentioned in section 1.2.

As per the steps mentioned in section 1.2, we chose a suitable SC problem to be our case study (SFS) and identified the scientific partners. We interviewed the scientists before the development of SFS to learn about their expertise and exposure to software skills. Then, we chose the SE approaches from the literature, suitable to SFS. A brief review of the literature about SE for SC is given in chapter 2. The SE practices which we followed during this case study are listed below.

- We followed a document driven approach in combination with a "faked" rational design process. In chapter 3, we discuss this approach in detail and in chapter 4, we present our observations while implementing this approach.
- To review the documents produced, we adopted "Task-based inspection". The details of this approach are discussed in chapter 3.
- We followed principles such as "Design for change" which deals with designing the software to accommodate anticipated changes. This is discussed in detail in chapters 3, 4 and 5.
- We used Git and an issue tracker for version control and work flow management. We also used the issue tracker for task-based inspection and communication between the scientists. This is elaborated in chapter 3 and chapter 4.
- Finally, we used regression testing to test our implementation. This is essential because in an SC domain, changes are inevitable. To accommodate 'trial and error' experiments, it is essential to ensure that any changes made do not break anything that was previously working. This approach is discussed in chapter 3 and chapter 4.

In applying the above mentioned SE principles and practices for the development of SFS, we had few difficulties in working with the scientists. We present these difficulties in the form of learnings and observations, which is discussed in chapter 4. Most of our observations resembled the literature and only a couple of them differed. We also identified that one of the observations, centering around the thought "upfront requirements are not feasible for SC" was a myth. Based on our case study, we identified that, with proper abstraction and using "faked document driven approach", it is possible to specify requirements in the beginning itself. This is elaborately discussed in chapter 5. The list of all observations are categorized and listed below.

1. **Observations that agree with the existing literature:** a) working on something that the scientists are interested in is not enough to promote SE practices, b) maintainability is a secondary consideration for scientific partners, c) scientists are hesitant to learn SE practices, d) verification and validation are challenging in SC, e) scientists naturally follow agile methodologies, f) common ground for communication has always been a problem, g) an interdisciplinary team is essential, h) scientists tend to choose programming language based on their familiarity, i) scientists prefer to use plots to visualize, verify and understand their science, j) early identification of test cases is advantageous, k) scientists have a positive attitude toward issue trackers, l) SC software should be designed for change, m) faking a rational design process for documentation is advisable for SC, n) Scientists prefer informal, collegial knowledge transfer, to reading documentation,

2. **Observations that disagree with the existing literature:** a) When unexpected results were obtained, our scientists chose to change the numerical algorithms, rather than question their scientific theories, b) Documentation of up-front requirements is feasible for SC

Following the steps mentioned in the methodology section (1.2) of chapter 1, we wanted to study the attitude of the scientists in applying SE to SC. To understand their attitudes, we interviewed the scientists after the development of the SFS and analyzed their feedback, to see if there are any changes in their attitude after witnessing the application of SE for SC. Our pre-development interview was primarily focused on understanding the background, specialization and their exposure to software skills. We found that some of them had limited exposure to coding but none of them were familiar with SE principles and practices. Most of our interviewees thought that software development, only involves coding. However, we saw a significant change in their attitudes after witnessing the application of SE into SC. Most of them accepted that SE can improve the quality of SC. The results from the interview and the feedback from the scientists are presented in chapter 6. This has reinforced the author's intuition in the article [74], a predecessor to this case study, that involving the scientists in the development process will improve the knowledge transfer of SE to scientific partners. In our case, the scientific partners appreciated the document driven design and also consider it as highly beneficial. However, it is also important to consider that some of the interviewees have been working with us since the beginning of this project and hence there is a possibility of an unconscious bias in their feedback. Also, some of the questions in the interview, even though unintentionally, were biased and may have led the scientists towards positive comments.

This thesis will act like a model document for those who want to apply SE for SC. It portrays the difficulties we encountered during the development of a SC application and offers solution from SE to overcome them. While developing SFS, we realized that scientists were not actually interested in reviewing documents and learning SE tools and principles. One exception to this is the issue tracker, which scientist readily adopted. Furthermore, communication between software developers and scientific partners was also a problem because of the scientific terminologies and software engineering jargons.

To address the above-mentioned difficulties, we adopted a "task-based inspection" to minimize the effort and time commitment required for document review. We also tried to stay away from SE jargons in our communication with the scientific partners during the post development interview. To motivate the scientists to apply SE for SC, we involved them from the beginning in the software development process as suggested in the literature. These approaches were successful and there was a positive change in the attitude of the scientists towards SE practices after the development of SFS. This work also insists that documents, templates and good engineering alone are not enough for the scientists to apply SE for SC. Tools, such as issue trackers, are important for the scientists to readily adopt SE approaches. In other words, for successful application of SE in SC, the SE tools and practices needs to be "lazy proof", that are easy to learn and use, without requiring much effort. Based on our experience in developing SFS, we listed a set of guidelines in appendix F for software developers developing SC applications.

7.2 Future work

In this section we present some of the future works and recommendations in regards to our case study. This is organized in two (2) categories as shown below.

7.2.1 Future works related to SFS

Some potential future work related to SFS are mentioned below.

- SFS was basically developed to predict the fraction solid with respect to temperature or time for a $1D$ heat transfer system. However, it could also be extended to $2D$ systems.

- The Piecewise module fits the input data by optimizing the fit based on an initial guess for break points. Currently, the initial guess is a manual input.

This could be automated and compute the initial guess for the breakpoints by detecting sudden change in slope.

- The piecewise module uses 3 sections to fit the input data. However, it needs to be automated to increase or decrease the number of sections and accommodate the fitting of input data without manual intervention.
- While developing SFS, we tried to automatically detect the phase change points. However, we could not accomplish this due to lack of time and resources. This could be considered as a future work for SFS.
- The current version of SFS requires the user to manually input the material properties of the alloy. This can be modified in the future by interfacing with other commercial software such as Thermocalc, which will predict the material properties based on the composition of the alloy.

7.2.2 Future works related to the case study

Some of the future works in relation to this case study of applying SE practices and principles to improve the quality of SC applications are listed below.

- In future, for developing a SC application, the system tests and metamorphic relations in the data must be identified earlier. Early identification of the system tests gives a chance to perform meaningful trial and error experiments when necessary. Metamorphic relations in the data, to some extent, may be used to validate the input and output data.
- When SFS did not produce expected results, the primary cause for the anomalies in the results was identified to be the input data. This is discussed in detail in chapter 4. In the future, this situation can be avoided by early identification of the system and data constraints. These constraints can be used to test the input data using metamorphic testing which is used to test for patterns in input data and warning messages may be generated for discrepancies or anomalies in the data. This will also improve the quality of the test cases.
- Throughout the development of SFS, we felt the need to collaborate with a domain scientist who could be a partner in code development. This will enable better knowledge transfer across scientist and developer. Such collaborations can be tried in future to evaluate the benefits.

The future works discussed in section 7.2.1 can be viewed as an extension to the existing software to make it more robust and complete. The SFS, in its current version must be viewed as a proof of concept and additional work is necessary to make it more reliable and robust. We also presented the future works in relation to our case study in section 7.2.2. This can also be viewed as the list of things we want to try if given a chance to develop SFS once again in addition to our existing approaches.

7.2.3 Recommendations for Scientists developing software

In this section, we provide a set of recommendations to scientists who want to develop a software, which will be used for a longer period of time and by person other than the developer.

1. Identify the critical requirements and the likely changes.
2. Identify the metamorphic relations in input and output data and the properties of correct solution.
3. Design the software encapsulating each likely change in a module.
4. Choose a programming language based on the requirements of the project and not based on familiarity.
5. Use suitable tools from SE such as Issue tracker to manage the work flow and communication.
6. Concentrate on verification and validation since the beginning of the project.
7. If trial and error experiments are necessary, choose proper approach and metrics to measure each trial.
8. If possible, collaborate with a software developer for better skill transferability.

Bibliography

[1] Jim Woodcock A. Shaon and E. Conway. Tools and guidelines for preserving and accessing software as a research output report ii: Case studies. technical report. the university of york., 2009.

[2] K. S. Ackroyd, S. H. Kinder, G. R. Mant, M. C. Miller, C. A. Ramsdale, and P. C. Stephenson. Scientific software development at a research facility. *IEEE Software*, 25(4):44–51, July 2008.

[3] United States Environmental Protection Agency. Models, tools, and databases for climate change research. `https://www.epa.gov/climate-research/models-tools-and-databases-climate-change-research`.

[4] Zeeshan Ahmed and Saman Zeeshan. Cultivating software solutions development in the scientific academia. *Recent Patents on Computer Science*, 7(1):54–66, 2014. `http://www.eurekaselect.com/node/122718/article`.

[5] Kieran Alden and Mark Read. Scientific software needs quality control. *Nature*, 502:448 EP –, 10 2013. `http://dx.doi.org/10.1038/502448d`.

[6] David H. Bailey, Jonathan M. Borwein, and Victoria Stodden, 2016.

[7] F. Benureau and N. Rougier. Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions. *ArXiv e-prints*, August 2017.

[8] Jeffrey C. Carver and Lorin Hochstein. observations about software development for high end computing, 2006.

[9] Jeffrey C. Carver, Neil P. Chue Hong, and George K. Thiruvathukal, editors. *Software Engineering for Science*. Chapman & Hall/CRC Computational Science. Chapman and Hall/CRC, 2016.

[10] 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 *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 550–559, Washington, DC, USA, 2007. IEEE Computer Society.

[11] Parmit K. Chilana, Carole L. Palmer, and Andrew J. Ko. Comparing bioinformatics software development by computer scientists and biologists: An exploratory study. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, SECSE '09, pages 72–79, Washington, DC, USA, 2009. IEEE Computer Society. http://dx.doi.org/10.1109/SECSE.2009.5069165.

[12] Christian Collberg, Todd Proebsting, and Alex M Warren. Repeatability and benefaction in computer systems research. Technical Report TR 14-04, Department of Computer Science, University of Arizona, Tucson, AZ, 2015. http://repeatability.cs.arizona.edu/v2/RepeatabilityTR.pdf.

[13] Tom Crick, Benjamin A. Hall, and Samin Ishtiaq. "Can I implement your algorithm?": A model for reproducible research software. *CoRR*, abs/1407.5981, 2014.

[14] A. P. Davison. Automated capture of experiment context for easier reproducibility in computational research. *Computing in Science & Engineering*, 14(4):48–56, July-Aug 2012.

[15] John B. Drake, Philip W. Jones, and Jr. George R. Carr. Overview of the software design of the community climate system model. *The International Journal of High Performance Computing Applications*, 19(3):177–186, 2005. https://doi.org/10.1177/1094342005056094.

[16] Anshu Dubey, Katie Antypas, Alan Calder, Bruce Fryxell, Don Lamb, Paul Ricker, Lynn Reid, Katherine Riley, Robert Rosner, Andrew Siegel, Francis Timmes, Natalia Vladimirova, and Klaus Weide. The software development process of flash, a multiphysics simulation code. In *Proceedings of the 5th International Workshop on Software Engineering for Computational Science and Engineering*, SE-CSE '13, pages 1–8, Piscataway, NJ, USA, 2013. IEEE Press. http://dl.acm.org/citation.cfm?id=2663370.2663372.

[17] S. M. Easterbrook and T. C. Johns. Engineering the software for understanding climate change. *Computing in Science Engineering*, 11(6):65–74, Nov 2009.

[18] Hans Fangohr, Maximilian Albert, and Matteo Franchin. Nmag micromagnetic simulation tool - software engineering lessons learned. *CoRR*, abs/1601.07392, 2016. http://arxiv.org/abs/1601.07392.

[19] S. Faulk, E. Loh, M. L. V. D. Vanter, S. Squires, and L. G. Votta. Scientific computing's productivity gridlock: How software engineering can help. *Computing in Science Engineering*, 11(6):30–39, Nov 2009. https://doi.org/10.1109/MCSE.2009.205.

[20] Jr. Frederick P. Brooks. *The Mythical Man-month (Anniversary Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 9 edition, 1995.

[21] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

[22] Gene H. Golub and James M. Ortega. *Scientific Computing and Differential Equations*. Academic Press, Boston, 1992. http://www.sciencedirect.com/science/article/pii/B9780080516691500034.

[23] Les Hatton. The t experiments: Errors in scientific software. *IEEE Computational Science and Engineering*, 4(2):27–38, April 1997. https://doi.org/10.1109/99.609829.

[24] Les Hatton and Andy Roberts. How accurate is scientific software? *IEEE Trans. Softw. Eng.*, 20(10):785–797, October 1994. http://dx.doi.org/10.1109/32.328993.

[25] Erika Check Hayden. Mozilla plan seeks to debug scientific code. https://www.nature.com/news/mozilla-plan-seeks-to-debug-scientific-code-1.13.

[26] Dustin Heaton and Jeffrey C. Carver. Claims about the use of software engineering practices in science. *Inf. Softw. Technol.*, 67(C):207–219, November 2015. http://dx.doi.org/10.1016/j.infsof.2015.07.011.

[27] Francisco Hernández, Purushotham Bangalore, and Kevin Reilly. Automating the development of scientific applications using domain-specific modeling. In *Proceedings of the Second International Workshop on Software Engineering for High Performance Computing System Applications*, SE-HPCS '05, pages 50–54, New York, NY, USA, 2005. ACM. http://doi.acm.org/10.1145/1145319.1145334.

[28] Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall S. Stanley. An overview of the trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, September 2005. http://doi.acm.org/10.1145/1089014.1089021.

[29] Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. `http://citeseer.ist.psu.edu/428727.html`.

[30] 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*, SECSE '09, pages 59–64, Washington, DC, USA, 2009. IEEE Computer Society. `http://dx.doi.org/10.1109/SECSE.2009.5069163`.

[31] IEEE. Recommended practice for software requirements specifications. *IEEE Std 830-1998*, pages 1–40, Oct 1998.

[32] John PA Ioannidis, David B Allison, Catherine A Ball, Issa Coulibaly, Xiangqin Cui, Aedín C Culhane, Mario Falchi, Cesare Furlanello, Laurence Game, Giuseppe Jurman, et al. Repeatability of published microarray gene expression analyses. *Nature genetics*, 41(2):149–155, 2009.

[33] Cezar Ionescu and Patrik Jansson. Dependently-Typed Programming in Scientific Computing — Examples from Economic Modelling. In *Revised Selected Papers of the 24th International Symposium on Implementation and Application of Functional Languages*, volume 8241 of *Lecture Notes in Computer Science*, pages 140–156. Springer International Publishing, 2012.

[34] ISO. Iso 25000 software product quality. `http://iso25000.com/index.php/en/iso-25000-standards/iso-25010`.

[35] Arne N. Johanson and Wilhelm Hasselbring. Software engineering for computational science: Past, present, future. *Computing in Science & Engineering*, Accepted:1–31, 2018.

[36] Upulee Kanewala and James M. Bieman. Testing scientific software: A systematic literature review. *CoRR*, abs/1804.01954, 2018. `http://arxiv.org/abs/1804.01954`.

[37] D. Kelly, D. Hook, and R. Sanders. Five recommended practices for computational scientists who write software. *Computing in Science and Engg.*, 11(5):48–53, September 2009. `http://dx.doi.org/10.1109/MCSE.2009.139`.

[38] Diane Kelly and Rebecca Sanders. The challenge of testing scientific software. In *Proceedings of the Conference for the Association for Software Testing*, pages 30–36, 2008.

[39] Diane Kelly and Terry Shepard. Task-directed software inspection technique: an experiment and case study. In *CASCON '00: Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative research*, page 6. IBM Press, 2000. http://portal.acm.org/citation.cfm?id=782040#.

[40] Diane Kelly and Terry Shepard. Eight maxims for software inspectors. *Software Testing, Verification and Reliability*, 14(4):243–256, 2004.

[41] Diane F. Kelly and Rebecca Sanders. Assessing the quality of scientific software. In *Proceedings of the First International Workshop on Software Engineering for Computational Science and Engineering (SECSE 2008)*, Leipzig, Germany, 2008. In conjunction with the 30th International Conference on Software Engineering (ICSE). http://www.cse.msstate.edu/~SECSE08/schedule.htm.

[42] Sarah Killcoyne and John Boyle. Managing chaos: Lessons learned developing software in the life sciences. *Computing in Science and Engg.*, 11(6):20–29, November 2009. http://dx.doi.org/10.1109/MCSE.2009.198.

[43] Bojana Koteska, Anastas Mishev, and Ljupco Pejov. Quantitative measurement of scientific software quality: Definition of a novel quality model. *International Journal of Software Engineering and Knowledge Engineering*, 28(03):407–425, 2018. https://doi.org/10.1142/S0218194018500146.

[44] Bojana Koteska, Anastas Mishev, and Ljupco Pejov. Quantitative measurement of scientific software quality: Definition of a novel quality model. *International Journal of Software Engineering and Knowledge Engineering*, 28(03):407–425, 2018. https://doi.org/10.1142/S0218194018500146.

[45] Charles W. Krueger. Software reuse. *ACM Comput. Surv.*, 24(2):131–183, June 1992. http://doi.acm.org/10.1145/130844.130856.

[46] Chemistry: LibreTexts. Liquid-solid phase diagrams: Tin and lead. https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supplemental_Modules_(Physical_and_Theoretical_Chemistry)/Equilibria/Physical_Equilibria/Liquid-Solid_Phase_Diagrams%3A_Tin_and_Lead.

[47] A M Lombardi. Seda: A software package for the statistical earthquake data analysis. *Scientific Reports*, 7:44171, 2017. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC5349582/.

[48] G. Uma Maheswari and Dr. V. V. Rama Prasad. Optimized software quality assurance model for testing scientific software.

[49] Daniel D. McCracken and Michael A. Jackson. Life cycle concept considered harmful. *SIGSOFT Softw. Eng. Notes*, 7(2):29–32, April 1982. http://doi.acm.org/10.1145/1005937.1005943.

[50] P. Messina. Gaining the broad expertise needed for high-end computational science and engineering research. *Computing in Science Engineering*, 17(2):89–90, Mar 2015.

[51] Reed Milewicz and Elaine M. Raybourn. Talk to me: A case study on coordinating expertise in large-scale scientific software projects. *CoRR*, abs/1809.06317, 2018.

[52] Greg Miller. A scientist's nightmare: Software problem leads to five retractions. *Science*, 314(5807):1856–1857, 2006. http://science.sciencemag.org/content/314/5807/1856.

[53] A. Nanthaamornphong, K. Morris, D. W. I. Rouson, and H. A. Michelsen. A case study: Agile development in the community laser-induced incandescence modeling environment (cliime). In *2013 5th International Workshop on Software Engineering for Computational Science and Engineering (SE-CSE)*, pages 9–18, May 2013.

[54] National Institute of Standards and Technology. Nist assesses technical needs of industry to improve software-testing. http://www.abeacha.com/NIST_press_release_bugs_cost.htm.

[55] D. L. Parnas. A technique for the specification of software modules with examples. *CACM*, 15(5):330–336, 1972.

[56] David L. Parnas and P.C. Clements. Software carpentry, 2006.

[57] 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.

[58] D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.

[59] Roger D. Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011.

[60] D. E. Post and L. G. Votta. Computational Science Demands a New Paradigm. *Physics Today*, 58(1):35–41, January 2005.

[61] Douglass E. Post and Richard P. Kendall. Software project management and quality engineering practices for complex, coupled multiphysics, massively parallel computational simulations: Lessons learned from asci. *IJHPCA*, 18:399–416, 2004.

[62] Michael Rilee and Thomas Clune. Towards test driven development for computational science with pfunit. In *Proceedings of the 2Nd International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, SE-HPCCSE '14, pages 20–27, Piscataway, NJ, USA, 2014. IEEE Press. https://doi.org/10.1109/SE-HPCCSE.2014.5.

[63] Patrick J. Roache. *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers, Albuquerque, New Mexico, 1998.

[64] Christopher Roy. Practical software engineering strategies for scientific computing. In *19th AIAA Computational Fluid Dynamics*. American Institute of Aeronautics and Astronautics, 2018/09/27 2009. https://doi.org/10.2514/6.2009-3997.

[65] U. Rüde, K. Willcox, L. McInnes, and H. Sterck. Research and education in computational science and engineering. *SIAM Review*, 60(3):707–754, 2018. https://doi.org/10.1137/16M1096840.

[66] R. Sanders and D. Kelly. Dealing with risk in scientific software development. *IEEE Software*, 25(4):21–28, July 2008.

[67] Judith Segal. When software engineers met research scientists: A case study. *Empirical Softw. Engg.*, 10(4):517–536, October 2005.

[68] Judith Segal. Some problems of professional end user developers. In *VLHCC '07: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 111–118, Washington, DC, USA, 2007. IEEE Computer Society.

[69] Judith Segal. Models of scientific software development, 2008. http://www.cs.ua.edu/~SECSE08/Papers/Segal.pdf.

[70] Judith Segal. Scientists and software engineers: a tale of two cultures, 2008. http://oro.open.ac.uk/17671/1/PPIG_08Segal.pdf.

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

[72] W. A. Simm, F. Samreen, R. Bassett, M. A. Ferrario, G. Blair, J. Whittle, and P. J. Young. Se in es: Opportunities for software engineering and cloud computing in environmental science. In *2018 ACM/IEEE 40th International Conference on Software Engineering: Software Engineering in Society*, pages 1–10, 2018.

[73] W. Spencer Smith. A rational document driven design process for scientific computing software. In Jeffrey C. Carver, Neil Chue Hong, and George Thiruvathukal, editors, *Software Engineering for Science*, Chapman & Hall/CRC Computational Science, chapter Examples of the Application of Traditional Software Engineering Practices to Science, pages 33–63. Taylor & Francis, Boca Raton, FL, 2016.

[74] W. Spencer Smith, Thulasi Jegatheesan, and Diane F. Kelly. Advantages, disadvantages and misunderstandings about document driven design for scientific software. In *Proceedings of the Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering (SE-HPCCE)*, November 2016. 8 pp.

[75] W. Spencer Smith and Nirmitha Koothoor. A document-driven method for certifying scientific computing software for use in nuclear safety analysis. *Nuclear Engineering and Technology*, 48(2):404 – 418, 2016. `http://www.sciencedirect.com/science/article/pii/S1738573315002582`.

[76] W. Spencer Smith, Nirmitha Koothoor, and Ned Nedialkov. A document driven method for facilitating certification of scientific computing software. *IEEE Transactions on Software Engineering*, Submitted 2014.

[77] W. Spencer Smith and Lei Lai. A new requirements template for scientific computing. In J. Ralyté, P. Ȧgerfalk, 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.

[78] 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(1):83–107, February 2007.

[79] W. Spencer Smith, Yue Sun, and Jacques Carette. Comparing psychometrics software development between CRAN and other communities. Technical Report CAS-15-01-SS, McMaster University, January 2015. 43 pp.

[80] W. Spencer Smith and Wen Yu. A document driven methodology for improving the quality of a parallel mesh generation toolbox. *Advances in Engineering Software*, 40(11):1155–1167, November 2009.

[81] Ian Sommerville, editor. *Software Engineering*. Pearson, 9th edition, 2011.

[82] Tim Storer. Bridging the chasm: A survey of software engineering practice in scientific programming. *ACM Comput. Surv.*, 50(4):47:1–47:32, August 2017.

[83] Burak Turhan, Lucas Layman, Madeline Diep, Forrest Shull, and Hakan Erdogmus, editors. *Making Software*. O'Reilly Media, 9th edition, 2010.

[84] Noel Viehmeyer. Waterfall and why itâĂŹs not suitable for software development, 2015. https://www.boost.co.nz/blog/2015/10/waterfall-and-why-its-not-suitable-for-software-development.

[85] Wikipedia. Haley industries. https://en.wikipedia.org/wiki/Haley_Industries.

[86] Wikipedia. Institute of electrical and electronics engineers. https://en.wikipedia.org/wiki/Institute_of_Electrical_and_Electronics_Engineers.

[87] From Wikipedia. Goodness of fit. https://en.wikipedia.org/wiki/Goodness_of_fit.

[88] Greg Wilson. Techblog: Git: The reproducibility tool scientists love to hate, 2018.

[89] 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. Plumbley, Ben Waugh, Ethan P. White, and Paul Wilson. Best practices for scientific computing. *PLoS Biol*, 12(1):e1001745, January 2014.

[90] Gregory V. Wilson. Software carpentry: lessons learned [version 1; referees: 3 approved], 2014.

[91] D. Woollard, N. Medvidovic, Y. Gil, and C. A. Mattmann. Scientific software as workflows: From discovery to distribution. *IEEE Software*, 25(4):37–43, July 2008.

Appendix A

Presentation: A Case Study to Develop Scientific Software

SFS: Software for Fraction Solid

- A case study to develop Scientific Software

Agenda

- Question (5 min)
- Presentation (40 min)
- Discussion (25 min)
- Questionnaire (15 min)

Scenario

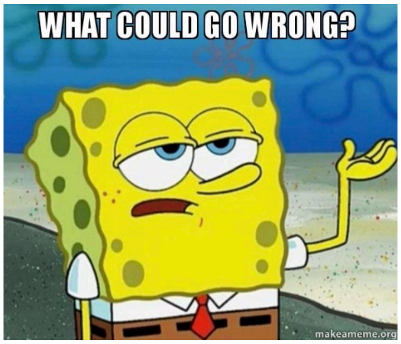
SFS is developed

Not expected output

Let's debug?

Question

Not expected output



Answer

1. Science
2. Code

We will pick the answer later ☺

Presentation

“No Science please !!”

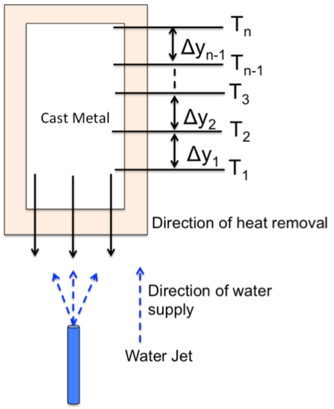
Terminologies

Casting is a process of pouring a molten metal into a mold.

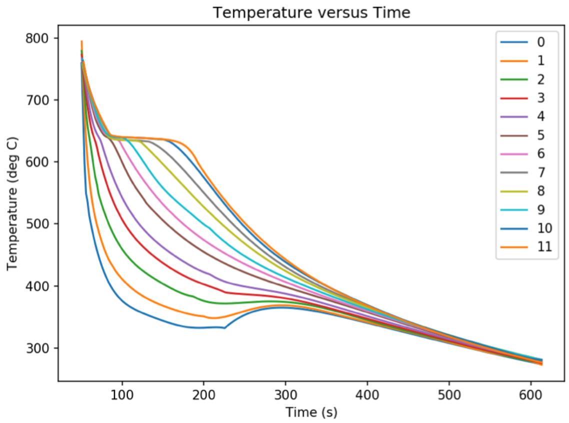
Solidification is a process by which the liquid metal solidifies into solid by losing heat.

Defects during solidification is related to solid fraction. - Expensive

SFS Experimental setup



Data



How SFS works?

Input:
Experiment details: H, D, n_TC, y_loc, dt, dy, Temp and time
Material properties: Cp, α, L and ρ

Solve $f_s(t)$ at location y^* such that the following ODE is satisfied with $f_s(t_L) = 0$:

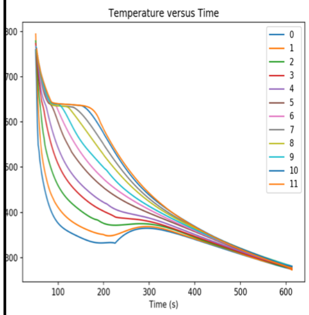
$$\dot{f}_s(f_s, t) = \frac{C_v(f_s)}{L\rho(f_s)}\left[\frac{\partial T(t)}{\partial t} - \alpha(f_s)\frac{\partial^2 T(t)}{\partial y^2}\right] \text{ where}$$

$$C_v(f_s) = C_v^b(1 - f_s) + C_v^e f_s$$
$$\rho(f_s) = \rho_b(1 - f_s) + \rho_e f_s$$
$$\alpha(f_s) = \alpha_b(1 - f_s) + \alpha_e f_s$$

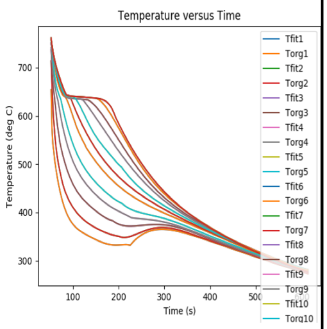
where $C_v^b = C_v^L(T_L)$, $C_v^e = C_v^S(T_S)$, $\rho_b = \rho_L(T_L)$, $\rho_e = \rho_S(T_S)$.
After solving the ODE, $f(t)$ and $T(t)$ are known, which means that $f(T)$ can be shown.

Output: fs(t)

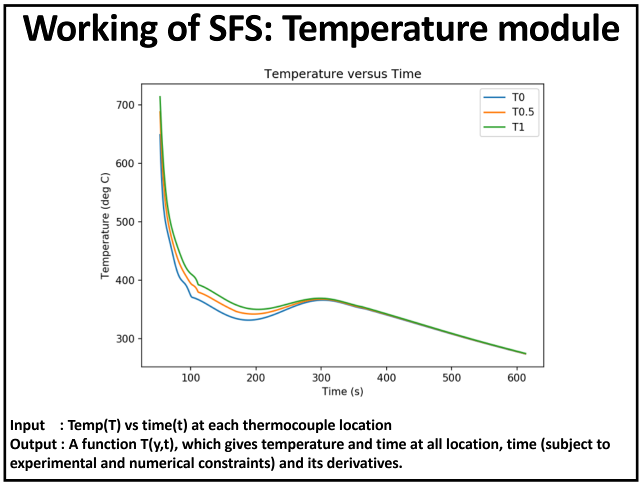
Working of SFS: Piecewise Module



Input Data: T vs t at each thermocouple location

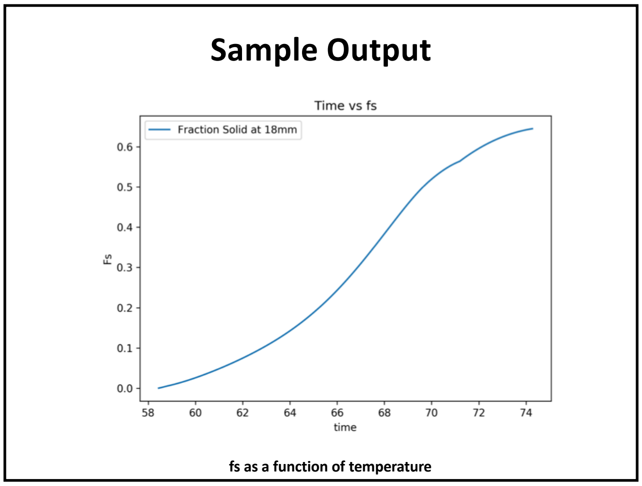


Output: Piecewise fit of each TC



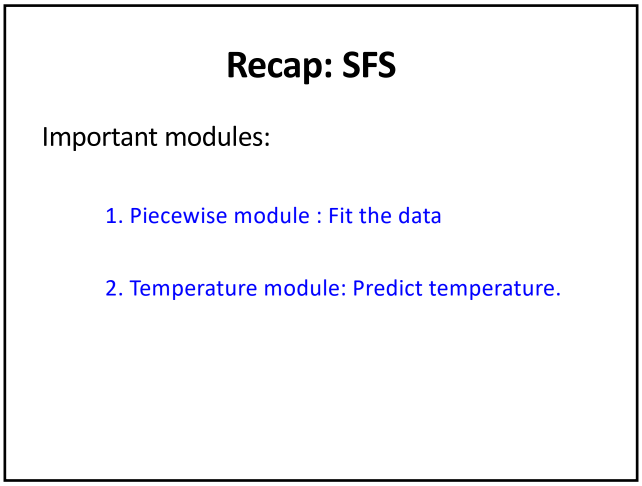
Working of SFS: Temperature module

Input : Temp(T) vs time(t) at each thermocouple location
Output : A function T(y,t), which gives temperature and time at all location, time (subject to experimental and numerical constraints) and its derivatives.



Sample Output

fs as a function of temperature

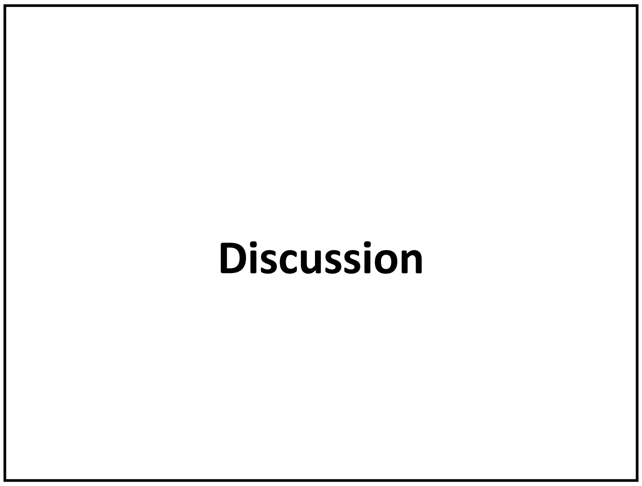


Recap: SFS

Important modules:

1. Piecewise module : Fit the data

2. Temperature module: Predict temperature.



Discussion

What if….

1. I graduate and don't give you any files? [Hypothetical ☺]
2. No documentation
3. Problem in the results: (First question)
4. Needs changes

SFS

How will you make changes to SFS?

- Different physical model?
- Integration with thermocalc?
- Changes in underlying mathematics (Splines instead of interpolation and regression)
- Changes in ODE

Steps

To illustrate the steps,

- ACTIVITY

Design for Change

Lets say we want to make a change,

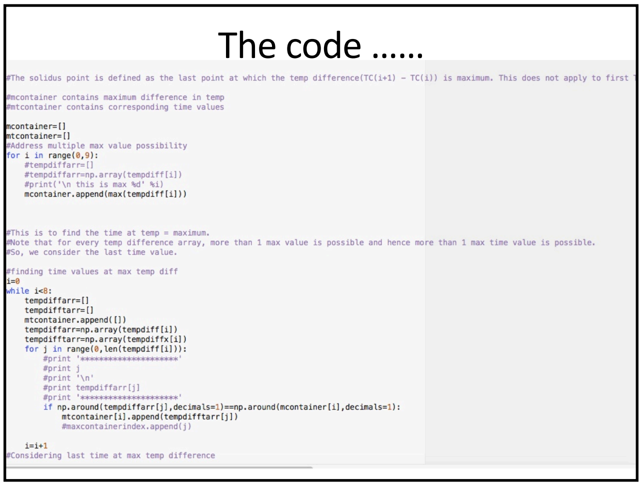
Task: Change the formula of dT/dt

Current formula:
(Forward difference) $\frac{T(y,t+\Delta t)-T(y,t)}{\Delta t}$

Proposed change:
(Central difference) $\frac{T(y,t+\Delta t)-T(y,t-\Delta t)}{2\Delta t}$

Lets go to code,

The code ……



```
#The solidus point is defined as the last point at which the temp difference(TC(i+1) - TC(i)) is maximum. This does not apply to first T

#mcontainer contains maximum difference in temp
#mtcontainer contains corresponding time values

mcontainer=[]
mtcontainer=[]
#Address multiple max value possibility
for i in range(0,9):
    #tempdiffarr=[]
    #tempdiffarr=np.array(tempdiff[i])
    #print('\n this is max %d' %i)
    mcontainer.append(max(tempdiff[i]))



#This is to find the time at temp = maximum.
#Note that for every temp difference array, more than 1 max value is possible and hence more than 1 max time value is possible.
#So, we consider the last time value.

#finding time values at max temp diff
i=0
while i<8:
    tempdiffarr=[]
    tempdifftarr=[]
    mtcontainer.append([])
    tempdiffarr=np.array(tempdiff[i])
    tempdifftarr=np.array(tempdiffx[i])
    for j in range(0,len(tempdiff[i])):
        #print '********************'
        #print j
        #print '\n'
        #print tempdiffarr[j]
        #print '********************'
        if np.around(tempdiffarr[j],decimals=1)==np.around(mcontainer[i],decimals=1):
            mtcontainer[i].append(tempdifftarr[j])
            #maxcontainerindex.append(j)

    i=i+1
#Considering last time at max temp difference
```

Please think about the following..

- Is code alone enough?
- Is the code easy to understand?
- Will it suffice if I explain everything now?
- Will you be able to modify the code for changes?

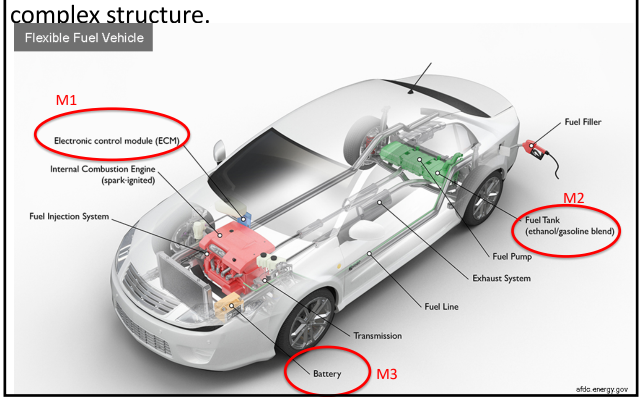
Answer: NO

Need more guidance?

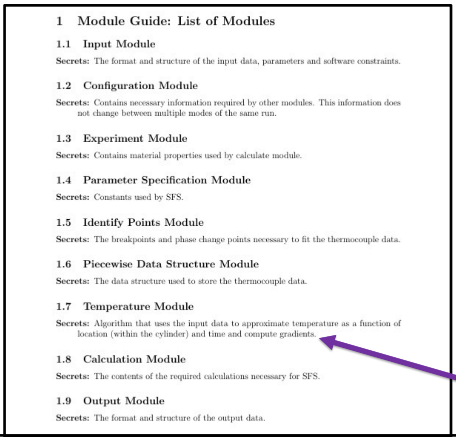
Module Guide

What is a module?

Module is an independent unit used to construct a complex structure.



Modules List



1 Module Guide: List of Modules

1.1 Input Module

Secrets: The format and structure of the input data, parameters and software constraints.

1.2 Configuration Module

Secrets: Contains necessary information required by other modules. This information does not change between multiple modes of the same run.

1.3 Experiment Module

Secrets: Contains material properties used by calculate module.

1.4 Parameter Specification Module

Secrets: Constants used by SFS.

1.5 Identify Points Module

Secrets: The breakpoints and phase change points necessary to fit the thermocouple data.

1.6 Piecewise Data Structure Module

Secrets: The data structure used to store the thermocouple data.

1.7 Temperature Module

Secrets: Algorithm that uses the input data to approximate temperature as a function of location (within the cylinder) and time and compute gradients.

1.8 Calculation Module

Secrets: The contents of the required calculations necessary for SFS.

1.9 Output Module

Secrets: The format and structure of the output data.

Is this easier to understand than the code?

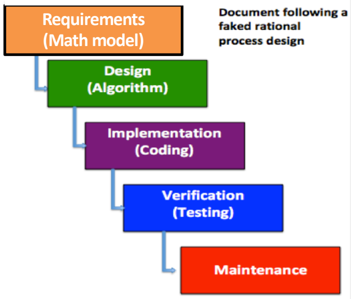
Why is it easy?

“Abstraction”

(Hiding complexities for ease of understanding and utility)

So, what we did?

Ideal Development process



What we did?

1. Start writing code from the beginning.
2. Prepare documents as if you followed a document driven approach.

Faked Rational process Design, Document driven approach

Next Question:

Did we finish our task??

Lets change the code….

So where do you explain code??

-MIS

(Module Interface Specification)

What is an interface?

Service: electricity

Algorithm: May be solar, wind or water

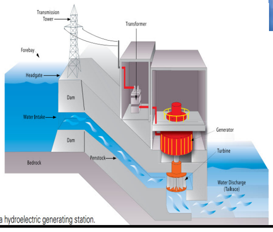
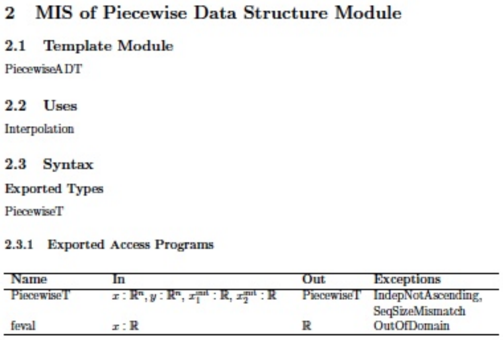


Diagram of a hydroelectric generating station.

MIS of a Module

- Variables
- Methods
- Algorithm

MIS of Piecewise module



2 MIS of Piecewise Data Structure Module

2.1 Template Module

PiecewiseADT

2.2 Uses

Interpolation

2.3 Syntax

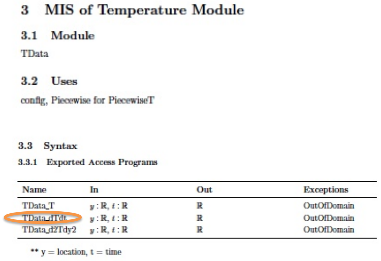
Exported Types

PiecewiseT

2.3.1 Exported Access Programs

Name	In	Out	Exceptions
PiecewiseT	$x : \mathbb{R}^n, y : \mathbb{R}^n, x_1^{init} : \mathbb{R}, x_2^{init} : \mathbb{R}$	PiecewiseT	IndepNotAscending, SeqSizeMismatch
feval	$x : \mathbb{R}$	$\mathbb{R}$	OutOfDomain

MIS of TData module



3 MIS of Temperature Module

3.1 Module

TData

3.2 Uses

config, Piecewise for PiecewiseT

3.3 Syntax

3.3.1 Exported Access Programs

Name	In	Out	Exceptions
TData_T	$y : \mathbb{R}, t : \mathbb{R}$	$\mathbb{R}$	OutOfDomain
TData_dTdt	$y : \mathbb{R}, t : \mathbb{R}$	$\mathbb{R}$	OutOfDomain
TData_d2Tdy2	$y : \mathbb{R}, t : \mathbb{R}$	$\mathbb{R}$	OutOfDomain

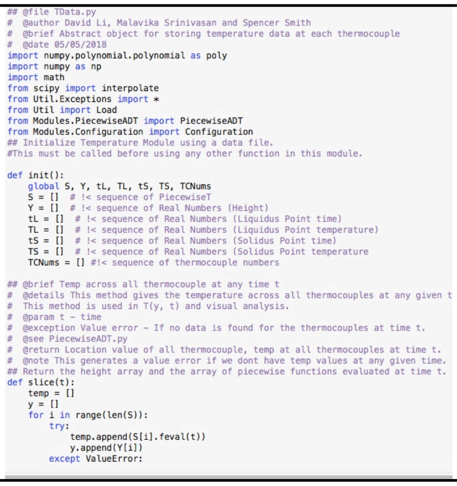
** y = location, t = time

Change

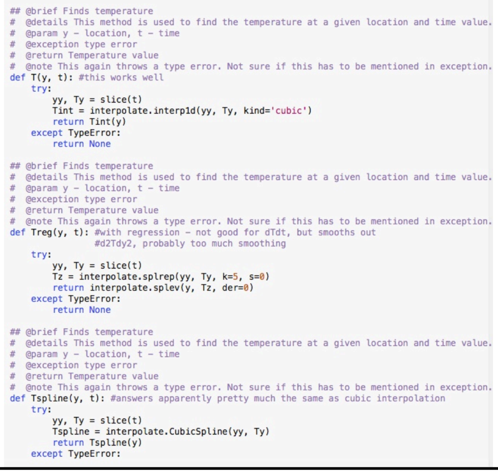
Lets change the formula…..

But, where is the code??

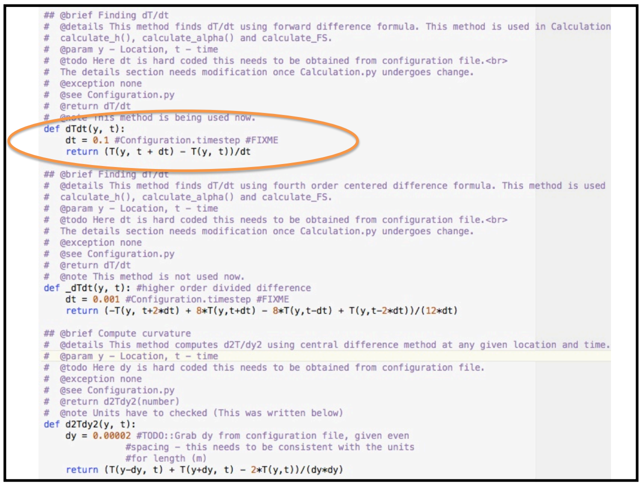
- The code.



```
## @file TData.py
#  @author David Li, Malavika Srinivasan and Spencer Smith
#  @brief Abstract object for storing temperature data at each thermocouple
#  @date 05/05/2018
import numpy.polynomial.polynomial as poly
import numpy as np
import math
from scipy import interpolate
from Util.Exceptions import *
from Util import Load
from Modules.PiecewiseADT import PiecewiseADT
from Modules.Configuration import Configuration
## Initialize Temperature Module using a data file.
#This must be called before using any other function in this module.

def init():
    global S, Y, tL, TL, tS, TS, TCNums
    S = []  # !< sequence of PiecewiseT
    Y = []  # !< sequence of Real Numbers (Height)
    tL = []  # !< sequence of Real Numbers (Liquidus Point time)
    TL = []  # !< sequence of Real Numbers (Liquidus Point temperature)
    tS = []  # !< sequence of Real Numbers (Solidus Point time)
    TS = []  # !< sequence of Real Numbers (Solidus Point temperature
    TCNums = [] #!< sequence of thermocouple numbers

## @brief Temp across all thermocouple at any time t
#  @details This method gives the temperature across all thermocouples at any given t
#  This method is used in T(y, t) and visual analysis.
#  @param t - time
#  @exception Value error - If no data is found for the thermocouples at time t.
#  @see PiecewiseADT.py
#  @return Location value of all thermocouple, temp at all thermocouples at time t.
#  @note This generates a value error if we dont have temp values at any given time.
## Return the height array and the array of piecewise functions evaluated at time t.
def slice(t):
    temp = []
    y = []
    for i in range(len(S)):
        try:
            temp.append(S[i].feval(t))
            y.append(Y[i])
        except ValueError:
```



```
## @brief Finds temperature
#  @details This method is used to find the temperature at a given location and time value.
#  @param y - location, t - time
#  @exception type error
#  @return Temperature value
#  @note This again throws a type error. Not sure if this has to be mentioned in exception.
def T(y, t): #this works well
    try:
        yy, Ty = slice(t)
        Tint = interpolate.interp1d(yy, Ty, kind='cubic')
        return Tint(y)
    except TypeError:
        return None

## @brief Finds temperature
#  @details This method is used to find the temperature at a given location and time value.
#  @param y - location, t - time
#  @exception type error
#  @return Temperature value
#  @note This again throws a type error. Not sure if this has to be mentioned in exception.
def Treg(y, t): #with regression - not good for dTdt, but smooths out
                #d2Tdy2, probably too much smoothing
    try:
        yy, Ty = slice(t)
        Tz = interpolate.splrep(yy, Ty, k=5, s=0)
        return interpolate.splev(y, Tz, der=0)
    except TypeError:
        return None

## @brief Finds temperature
#  @details This method is used to find the temperature at a given location and time value.
#  @param y - location, t - time
#  @exception type error
#  @return Temperature value
#  @note This again throws a type error. Not sure if this has to be mentioned in exception.
def Tspline(y, t): #answers apparently pretty much the same as cubic interpolation
    try:
        yy, Ty = slice(t)
        Tspline = interpolate.CubicSpline(yy, Ty)
        return Tspline(y)
    except TypeError:
```



```
## @brief Finding dT/dt
#  @details This method finds dT/dt using forward difference formula. This method is used in Calculation
#  calculate_h(), calculate_alpha() and calculate_FS.
#  @param y - Location, t - time
#  @todo Here dt is hard coded this needs to be obtained from configuration file.<br>
#  The details section needs modification once Calculation.py undergoes change.
#  @exception none
#  @see Configuration.py
#  @return dT/dt
#  @note This method is being used now.
def dTdt(y, t):
    dt = 0.1 #Configuration.timestep #FIXME
    return (T(y, t + dt) - T(y, t))/dt

## @brief Finding dT/dt
#  @details This method finds dT/dt using fourth order centered difference formula. This method is used
#  calculate_h(), calculate_alpha() and calculate_FS.
#  @param y - Location, t - time
#  @todo Here dt is hard coded this needs to be obtained from configuration file.<br>
#  The details section needs modification once Calculation.py undergoes change.
#  @exception none
#  @see Configuration.py
#  @return dT/dt
#  @note This method is not used now.
def _dTdt(y, t): #higher order divided difference
    dt = 0.001 #Configuration.timestep #FIXME
    return (-T(y, t+2*dt) + 8*T(y,t+dt) - 8*T(y,t-dt) + T(y,t-2*dt))/(12*dt)

## @brief Compute curvature
#  @details This method computes d2T/dy2 using central difference method at any given location and time.
#  @param y - Location, t - time
#  @todo Here dy is hard coded this needs to be obtained from configuration file.
#  @exception none
#  @see Configuration.py
#  @return d2Tdy2(number)
#  @note Units have to checked (This was written below)
def d2Tdy2(y, t):
    dy = 0.00002 #TODO::Grab dy from configuration file, given even
                 #spacing - this needs to be consistent with the units
                 #for length (m)
    return (T(y-dy, t) + T(y+dy, t) - 2*T(y,t))/(dy*dy)
```

Why so much effort needed?

"Design for change"

(Design a software anticipating changes)

Why we need it?

What if you want to fit a different data?

What if you do not want to use interpolation and regression?

Discussion

How do you know SFS modules are correct?

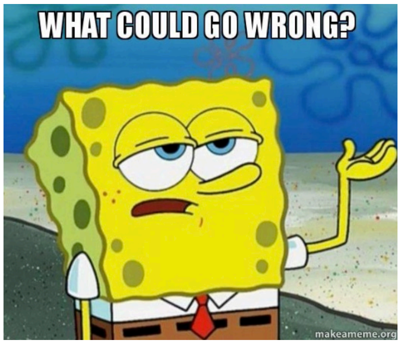
How do you know your change hasn't broken anything?

Confidence in code?

Testing

Question

Software fails



Answer

1. Science
2. Code

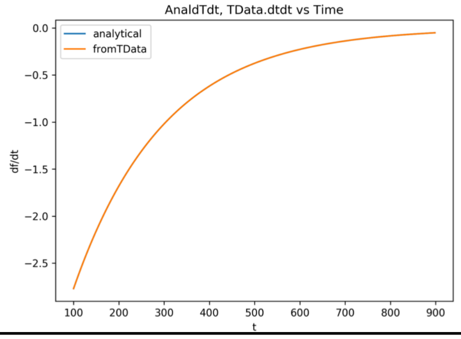
"What would be the first step to prove code is not wrong?"

Testing

Testing

Let us look at dT/dt…

(Before the formula change)



Testing

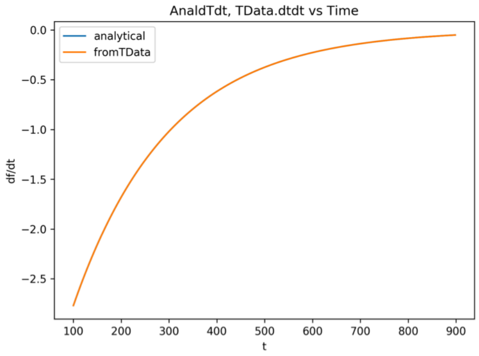
Pytest

Test case fails

Visual inspection can fail

Testing

(After the formula change)



Testing

Pytest

Test case passed

Testing

1. Visual inspections can fail

2. Manual testing too tedious

Solution: Regression Testing

Regression Testing

- **Regression testing** is re-running functional and non-functional tests to ensure that previously developed and tested software still performs after a change.
- Approximation error:
  – Absolute error
  – Relative error

Where is the science?

Where is the science documented?

SRS

Software Requirement specification

No information about design.

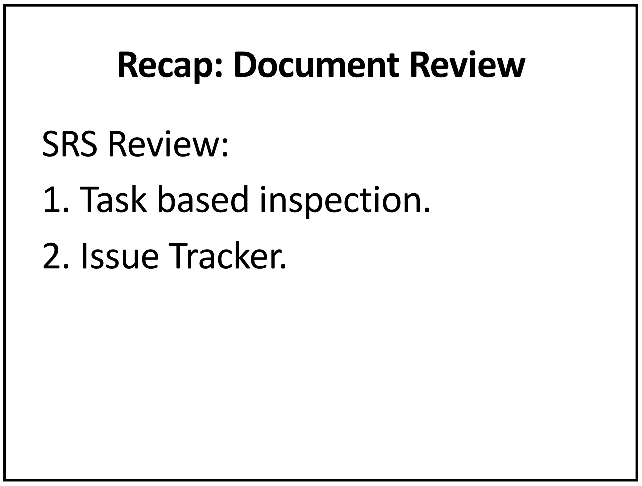
[Handout : Table of contents]

(Background, Goals, Requirements, Assumptions and equations to solve for SFS.)

Software Tools

- Issue Tracker

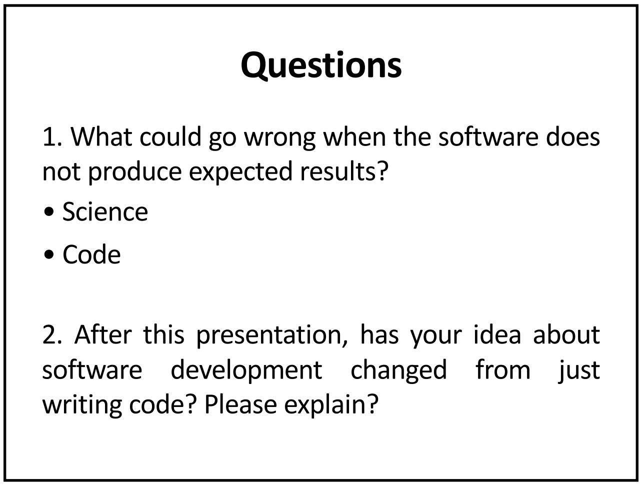
- doxygen



Recap: Document Review

SRS Review:

1. Task based inspection.
2. Issue Tracker.

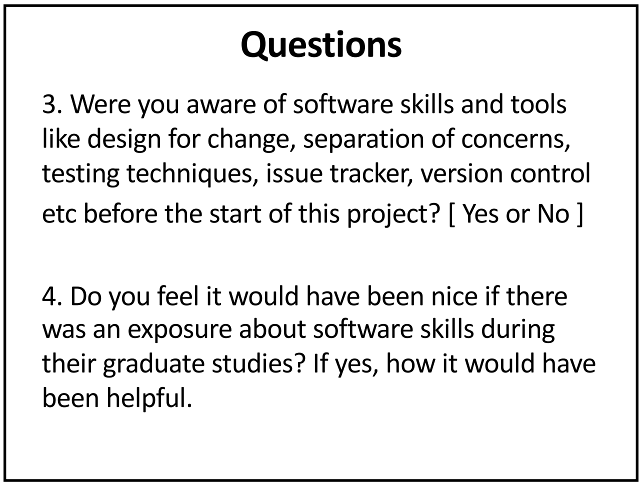


Questions

1. What could go wrong when the software does not produce expected results?

- Science
- Code

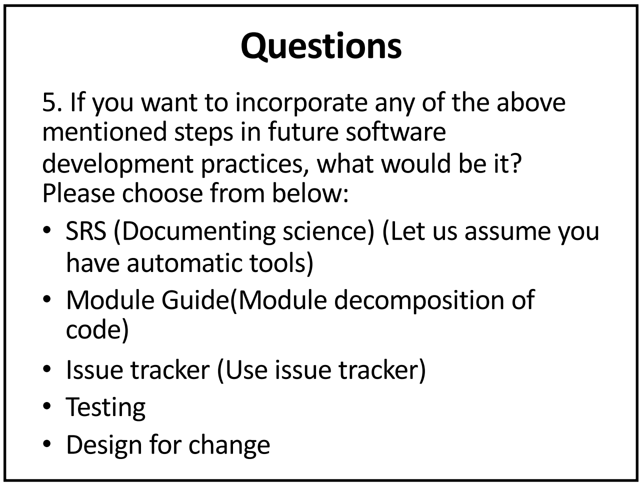
2. After this presentation, has your idea about software development changed from just writing code? Please explain?



Questions

3. Were you aware of software skills and tools like design for change, separation of concerns, testing techniques, issue tracker, version control etc before the start of this project? [Yes or No]

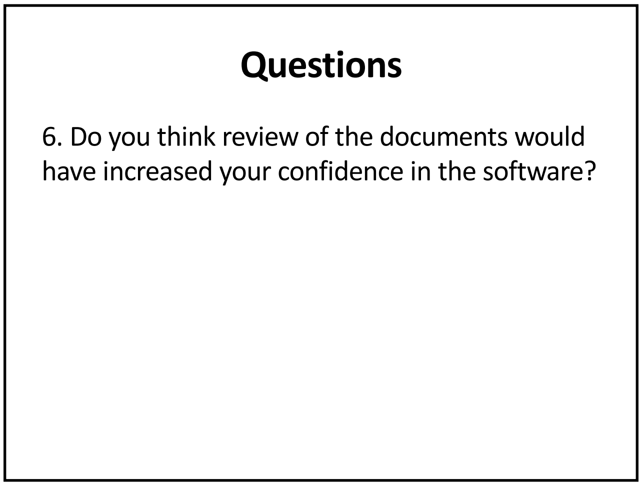
4. Do you feel it would have been nice if there was an exposure about software skills during their graduate studies? If yes, how it would have been helpful.



Questions

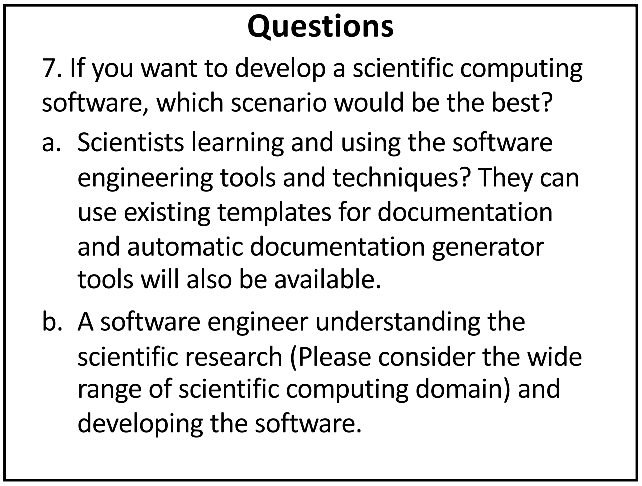
5. If you want to incorporate any of the above mentioned steps in future software development practices, what would be it? Please choose from below:

- SRS (Documenting science) (Let us assume you have automatic tools)
- Module Guide(Module decomposition of code)
- Issue tracker (Use issue tracker)
- Testing
- Design for change



Questions

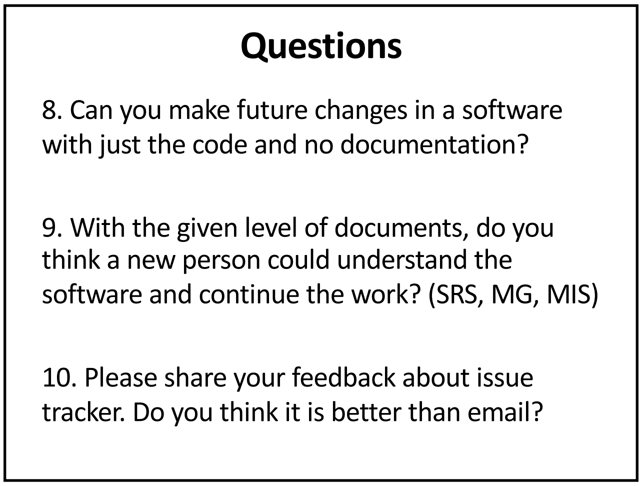
6. Do you think review of the documents would have increased your confidence in the software?



Questions

7. If you want to develop a scientific computing software, which scenario would be the best?

a. Scientists learning and using the software engineering tools and techniques? They can use existing templates for documentation and automatic documentation generator tools will also be available.
b. A software engineer understanding the scientific research (Please consider the wide range of scientific computing domain) and developing the software.



Questions

8. Can you make future changes in a software with just the code and no documentation?

9. With the given level of documents, do you think a new person could understand the software and continue the work? (SRS, MG, MIS)

10. Please share your feedback about issue tracker. Do you think it is better than email?

Thank you !!!

Appendix B

SRS: Software Requirement Specification

Software Requirements Specification for Software for Solidification

Malavika Srinivasan and Spencer Smith

December 13, 2018

Contents

1 Reference Material

This section records information for ease of reference. The information includes the units, symbols and abbreviations used in this document.

1.1 Table of Units

Throughout this document SI (Système International d'Unités) is employed as the unit system. (The one exception is that degrees Celsius is used for temperature, instead of Kelvin, in keeping with standard practice for the problem domain.) In addition to the basic units, several derived units are used as described below. For each unit, the symbol is given followed by a description of the unit with the SI name.

symbol	**unit**	**SI**
m	length	metre
kg	mass	kilogram
s	time	second
°C	temperature	centigrade
J	energy	Joule
W	power	Watt ($\mathrm{W} = \mathrm{J\,s^{-1}}$)

1.2 Table of Symbols

The table that follows summarizes the symbols used in this document along with their units. The choice of symbols was made to be consistent with the heat transfer literature and with existing documentation for software for solidification systems. When vector quantities are presented, the units apply for each element of the vector. The symbols are listed in alphabetical order.

symbol	**unit**	**description**
C_v	$\mathrm{J/(m^3\,°C)}$	Volumetric heat capacity
C_v^L	$\mathrm{J/(m^3\,°C)}$	Volumetric heat capacity at beginning of solidification
C_v^S	$\mathrm{J/(m^3\,°C)}$	Volumetric heat capacity at end of solidification
C_p	$\mathrm{J/(kg\,°C)}$	Specific heat capacity
f_s	no unit	Fraction of solid formed during solidification
$\dot{f}_s$	1/s	Fraction solid w.r.t. time across different locations
g	$\mathrm{W/m^3}$	Rate of volumetric heat generation
k	$\mathrm{W/(m\,°C)}$	Thermal conductivity

L	J/kg	Latent heat of solidification
m	kg	Mass
$\mathbf{q}$	W/m^2	The thermal flux vector
q_s	W/m^3	Heat released during solidification
Q	J	Latent heat energy
t	s	Time
T	°C	Temperature
$\dot{T}$	°C/s	Change of temperature w.r.t. time across different locations
V	m^3	Volume
α	m^2/s	Thermal diffusivity
α_S	m^2/s	Thermal diffusivity of solid as a function of temperature
α_b	m^2/s	Thermal diffusivity at beginning of solidification (at the liquidus point)
α_e	m^2/s	Thermal diffusivity when first solid (at the solidification point)
ρ	kg/m^3	Density
ρ_b	kg/m^3	Density at beginning of solidification
ρ_e	kg/m^3	Density at end of solidification
ρ_L	kg/m^3	Density in liquidus zone as a function of temperature
ρ_S	kg/m^3	Density in solidus zone as a function of temperature
τ	s	Temporary time variable
Δy	m	Distance between thermocouples
λ	variable	Generic material properties
∇	no unit	Gradient operator

1.3 Abbreviations and Acronyms

symbol	description
A	Assumption
DD	Data Definition
GD	General Definition
GS	Goal Statement
IM	Instance Model
LC	Likely Change
ODE	Ordinary Differential Equation
PS	Physical System Description
R	Requirement
SRS	Software Requirements Specification
SFS	Software for Solidification
T	Theoretical Model
1D	1 Dimensional
w.r.t.	with respect to

2 Introduction

Solidification is a branch of science that deals with the study of transition of a liquid, in the present case a liquid metal alloy, to a solid as its temperature is lowered. During solidification heat is given out from the liquid phase to form a solid. During this process, the element or the alloy undergoes phase transition from liquid to 2 phase (solid + liquid) and finally to solid. In this project, software is developed to estimate the fraction of solid present in the 2 phase zone. An experiment is devised with a cylinder of alloy such that unidirectional heat removal can be assumed. Data is collected, via thermocouples, of the temperature inside the cylinder over time. This data can then be used to characterize the solid fraction as a function of temperature and the rate of cooling.

The following section provides an overview of the Software Requirements Specification (SRS) for a solidification system for a cast material. The developed program will be referred to as Software For Solidification (SFS). This section explains the purpose of this document, the scope of the system, the organization of the document and the intended audience.

2.1 Purpose of Document

The main purpose of this document is to explain the physics behind SFS. The SRS is abstract because the contents say *what* problem is being solved, but do not say *how* to solve it. This document will be used as a starting point for subsequent development phases, including writing the design specification and the software verification and validation plan. The design document will show how the requirements are to be realized, including decisions on the numerical algorithms and programming environment. The verification and validation plan will show the steps that will be used to increase confidence in the software documentation and the implementation. Although the SRS fits in a series of documents that follow the so-called waterfall model, the actual development process is not constrained in any way. Even when the process is not waterfall, as Parnas and Clements [4] point out, the most logical way to present the documentation is still to "fake" a rational design process.

2.2 Scope of Requirements

The scope of the requirements is limited to predict the solidification characteristics of a specific alloy system. Given the appropriate inputs, the code for SFS is intended to predict the solid fraction of the solidifying alloy. This entire document is written assuming that the heat removal out of the system is unidirectional.

The scope of these requirements do not extend to the experimental data collection. SFS is not itself a data acquisition system. The data will be collected by another program and then provided to SFS.

2.3 Intended Audience

This document is intended for the users of this software as well as software engineers and programmers who may be working in this project. The typical reader is expected to have basic knowledge about thermodynamics, physics and mathematics. This document will be reviewed by scientists and mechanical engineers with respect to the theory behind the solidification of liquid alloys.

2.4 Organization of Document

The organization of this document follows the template for an Software Requirement Specification (SRS) for scientific computing software proposed by [3] and [6]. The presentation follows the standard pattern of presenting goals, theories, definitions, and assumptions. For readers that would like a more bottom up approach, they can start reading the instance model in Section 5.2.5 and trace back to find any additional information they require. The instance model provides the Ordinary Differential Equation (ODE) and algebraic equations that model the solidification process. SFS solves the ODE mentioned in the instance model.

The goal statements are refined to the theoretical models, and theoretical models to the instance models.

3 Background

Solidification is a branch of science that deals with the study of transition of a liquid to solid when its temperature is lowered below its melting or freezing point. Understanding solidification is critical for a casting process, where an object is made by pouring molten metal or molten alloy into a mold.

In an alloy, the components of the alloy may have different freezing points and may solidify at different temperatures. During this process, there are 3 different phases - Solid zone, Liquid zone and 2 phase zone (solid + liquid). Understanding solidification involves identifying the points at which the elements present in the alloy start and finish solidifying. These points are called liquidus, eutectic and solidus points. These points are marked in Figure 1. The liquidus point occurs at the temperature where solids first start to form. It is represented by a sudden change in curvature of the cooling curve. A eutectic point is when the rate of energy leaving the material matches the rate of energy entering through solidification. At the eutectic point the temperature remains constant until all of the material has solidified. The solidus temperature specifies the temperature below which a material is completely solid.

The main aim of SFS is to estimate the solid fraction in the 2 phase zone, which is the percentage of solid present in the mixture of solid and liquid. Finding the fraction solid is essential in predicting the properties of the solidified alloy. These properties determine the quality of the castings.

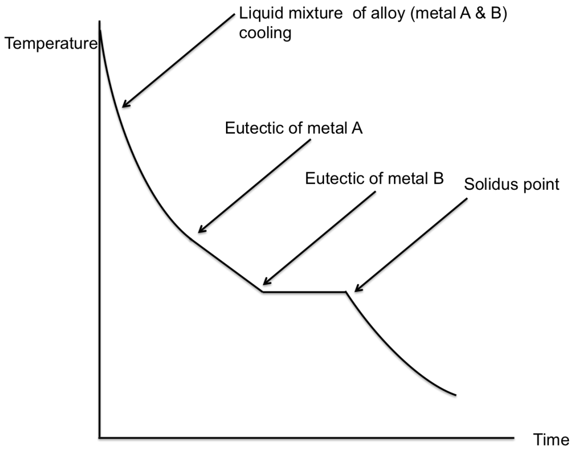


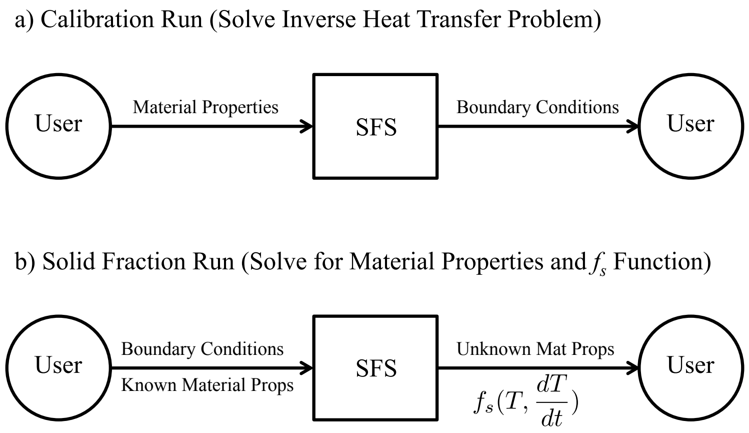
Figure 1: Typical cooling in a binary alloy with liquidus, solidus and eutectic points identified

4 General System Description

This section provides general information about the system, identifies the interfaces between the system and its environment, and describes the user characteristics and the system constraints.

4.1 System Context

Figure 2 shows the system context. A circle represents an external entity outside the software, the user in this case. A rectangle represents the software system itself (SFS). Arrows are used to show the data flow between the system and its environment.


a) Calibration Run (Solve Inverse Heat Transfer Problem)

b) Solid Fraction Run (Solve for Material Properties and f_s Function)

Figure 2: System Context

The system has three modes:

1. Configuration Mode: Many experiments will share the same configuration information, such as the dimensions of the cylinder and the locations of the thermocouples; therefore, the stable/persistent data is entered in a separate mode. This data entry only needs to be repeated when their are changes to the configuration.

2. Calibration Mode: An inverse heat transfer problem is solved to determine the thermal boundary conditions on the bottom of the plate using an experiment with an alloy that has known material properties. Specifically, the purpose is to find the heat transfer coefficient.

3. Solid Fraction Mode: Using the boundary conditions determined from the calibration run and the known material properties the system solves for the unknown material properties and for the solid fraction as a function of temperature and temperature gradient.

SFS is mostly self-contained. The only external interaction is through the user interface. The responsibilities of the user and the system are as follows:

- User Responsibilities:

- Enter configuration information
- Decide whether to do a Calibration Run or a Solid Fraction Run
- Run the appropriate experiment to obtain the required data
- Provide the input data to the system, ensuring no errors in the data entry
- Take care that consistent units are used for input variables

- SFS Responsibilities:
  - Detect data type mismatch, such as a string of characters instead of a floating point number
  - Determine if the inputs satisfy the required physical and software constraints
  - Calculate the required outputs

4.2 User Characteristics

The end user of SFS should have an understanding of level I calculus, thermodynamics and solidification.

4.3 System Constraints

SC1: The software needs to work on Windows. Ideally, it should also be portable to other operating systems, including Mac OSX and Linux.

5 Specific System Description

This section first presents the problem description, which gives a high-level view of the problem to be solved. This is followed by the solution characteristics specification, which presents the assumptions, theories, definitions and finally the instance models that models the solidification system.

5.1 Problem Description

SFS is a computer program developed to find the solid fraction as a function of temperature and rate of cooling.

5.1.1 Terminology and Definitions

This subsection provides a list of terms that are used in the subsequent sections and their meaning, with the purpose of reducing ambiguity and making it easier to correctly understand the requirements:

- Heat Flux: The rate of heat energy transfer per unit area.

- Specific Heat: Heat capacity per unit mass.
- Isotropic: The phenomena by which the physical property of the material has the same value when measured in different directions.
- Thermal conductivity: The rate at which heat passes through a specified material, expressed as the amount of heat that flows per unit time through a unit area with a temperature gradient of one degree per unit distance.
- Thermal diffusivity: Thermal conductivity divided by density and specific heat capacity.
- Advection: Transfer of heat or matter by the flow of a fluid, especially horizontally in the atmosphere.

5.1.2 Physical System Description

The physical system of SFS, as shown in Figure 3, includes the following elements:

PS1: Mold containing the cast metal.

PS2: Water jet under the mold to provide unidirectional heat extraction.

main

5.1.3 Goal Statements

For a given experiment with a metal alloy, using the thermocouple locations, temperature readings, material properties and initial conditions, SFS:

GS1: Computes the solid fraction (f_s) as a function of temperature and cooling rate ($f_s(T, \frac{dT}{dt})$).

5.2 Solution Characteristics Specification

The instance model (ODE) that govern SFS is presented in Subsection 5.2.5. The information to understand the meaning of the instance model and its derivation is also presented, so that the instance model can be verified.

5.2.1 Assumptions

This section simplifies the original problem and helps in developing the theoretical model by filling in the missing information for the physical system. The numbers given in the square brackets refer to the theoretical model [T], general definition [GD], data definition [DD], instance model [IM], likely change [LC], or unlikely change [UC], in which the respective assumption is used.

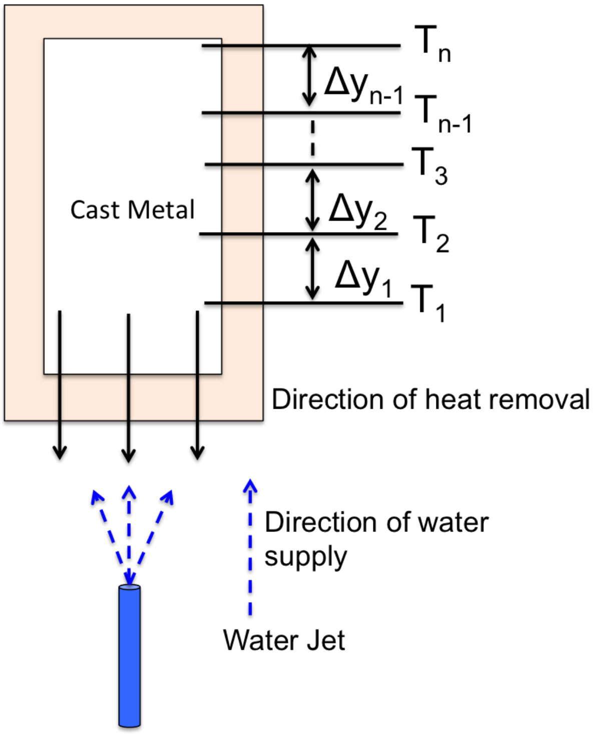


Figure 3: SFS system with unidirectional heat extraction, where n is the number of thermocouples, T_i is the thermocouple temperature at location i and Δy_i is the distance between the thermocouples i and $i+1$.

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, UC1].

A2: The heat removal is assumed to be unidirectional and the heat conduction in axial direction is assumed to be 0 [GD1, UC2].

A3: Heat transfer through the cylinder takes place by conduction only, not advection. [DD5].

A4: We assume that $C_v(T)$ can be expressed as a linear combination of the values at the beginning and at the end of solidification [IM4, DD1, GD2].

at

A5: We assume that $\alpha(T)$ can be expressed as a linear combination of the values at the beginning and at the end of solidification [IM4, DD2, GD2].

A6: We assume that $\rho(T)$ can be expressed as a linear combination of the values at the beginning and at the end of solidification [IM4, DD3, GD2].

A7: Thermal conductivity through the liquid and solid metal is isotropic.

A8: Newton's law of convective cooling applies between the water and the cast alloy [GD3, DD8].

A9: The heat transfer coefficient at the bottom of the cylinder is assumed to be independent of temperature [GD3, LC1].

A10: The thermal resistance due to the thermocouples is assumed to be negligible [GD4].

A11: The cast metal is perfectly insulated by the sand mold so that there is no heat loss from the sand mold [GD1].

A12: The density of the solidifying material is assumed to be constant for the derivation of $\dot{f}_s$ [IM4].

A13: The data collected from the thermocouples starts at time 0 and there is a constant time step between each data point [GD4, LC2].

5.2.2 Theoretical Models

This section focuses on the general equations and laws that SFS is based on.

Number	T1
Label	**Conservation of thermal energy**
Equation	$-\nabla \cdot \mathbf{q} + g = \rho C_p \frac{\partial T}{\partial t}$
Description	The above equation gives the conservation of energy for time varying heat transfer in a material of specific heat capacity C_p ($\mathrm{J\,kg^{-1}\,°C^{-1}}$) and density ρ ($\mathrm{kg\,m^{-3}}$) where $\mathbf{q}$ is the thermal flux vector ($\mathrm{W\,m^{-2}}$) g is the rate of volumetric heat generation ($\mathrm{W\,m^{-3}}$) T is the temperature (°C) t is time (s) ∇ is the gradient operator. For this equation to apply, other forms of energy, such as mechanical energy, are assumed to be negligible in the system (A1). In general, ρ and C_p depend on temperature T.
Source	http://www.efunda.com/formulae/heat_transfer/conduction/overview_cond.cfm
Ref. By	IM4, GD1, DD5

Number	T2
Label	**Latent heat energy**
Equation	$Q(t) = \int_a^b \frac{dQ(\tau)}{d\tau} d\tau$, with $Q(0) = 0$
Description	Q is the change in thermal energy (J), $\int_a^b \frac{dQ(\tau)}{d\tau} d\tau$ is the rate of change of Q with respect to time τ (s). a and b are the times (s) when calculating the latent heat energy starts and ends, respectively.
Source	http://en.wikipedia.org/wiki/Latent_heat
Ref. By	IM4, DD5

Number	T3
Label	**Fourier's Law**
Equation	$\mathbf{q} = -k\nabla T$
Description	An empirical relationship between the conduction rate in a material and the temperature gradient in the direction of energy flow. k is the thermal conductivity ($\mathrm{W\,m^{-1}\,°C^{-1}}$) $\mathbf{q}$ is the thermal flux vector ($\mathrm{W\,m^{-3}}$) ∇ is the gradient operator. T is the temperature (°C). To simplify Fourier's law to only have one thermal conductivity, the above equation assumes that the material is isotropic (A7).
Source	https://en.wikipedia.org/wiki/Thermal_conduction#Fourier.27s_law
Ref. By	GD1, IM4

5.2.3 General Definitions

This section collects the laws and equations that will be used in deriving the data definitions, which in turn will be used to build the instance models.

Number	GD1
Label	**Conservation of Thermal Energy in the Cylinder**
SI Units	$^{\circ}\mathrm{C}\,\mathrm{s}^{-1}$
Equation	$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial y^2} + \frac{q_s}{C_v}$, where $\alpha = \frac{k}{\rho C_p}$ and $C_v = \rho C_p$
Description	where α is the thermal diffusivity in $\mathrm{m}^2\,\mathrm{s}^{-1}$ C_v is the volumetric heat capacity in $\mathrm{J}\,\mathrm{m}^{-3}\,^{\circ}\mathrm{C}^{-1}$ ρ is the density in $\mathrm{kg}\,\mathrm{m}^{-3}$ C_p is the heat capacity in $\mathrm{J}\,\mathrm{kg}^{-1}\,^{\circ}\mathrm{C}^{-1}$ T is the temperature in °C t is the time in s k is the thermal conductivity in $\mathrm{W}\,\mathrm{m}^{-1}\,^{\circ}\mathrm{C}^{-1}$ q_s is the rate of heat generated by solidification in $\mathrm{W}\,\mathrm{m}^{-3}$. This equation is derived by substituting T3 in T1 and invoking A11 and A2, which makes $\nabla = [0, \frac{\partial}{\partial y}, 0]$, where y is the dimension along the axis of the cylinder. Also, g is relabelled as q_s, to clarify the connection that the heat is generated by solidification. To isolate $\frac{\partial T}{\partial t}$ on the left hand side of the equation, the conservation equation is divided by $\rho C_p = C_v$.
Source	–
Ref. By	IM4

Number	GD2
Label	**Generic Material Properties dependent on Temperature**
SI Units	Units of $\lambda(f_s)$
Equation	$\lambda(f_s) = \lambda_b(1 - f_s) + \lambda_e f_s$
Description	where λ is any generic material property which is dependent on solid fraction. λ_b and λ_e represent the material property at the beginning and end of solidification. f_s represents the fraction of solid formed. Based on assumptions A4, A5 and A6 material properties like C_v, ρ and α can be expressed as a linear combination of the values at the beginning and end of solidification.
Source	[1]
Ref. By	DD1, DD2, DD3

Number	GD3
Label	**Newton's law of cooling**
SI Units	$\mathrm{W\,m^{-2}}$
Equation	$q(t) = h\Delta T(t)$
Description	Newton's law of cooling describes convective cooling from a surface. The law is stated as: the rate of heat loss from a body is proportional to the difference in temperatures between the body and its surroundings. $q(t)$ is the thermal flux ($\mathrm{W\,m^{-2}}$). h is the heat transfer coefficient, assumed independent of T (A9) ($\mathrm{W\,m^{-2}\,°C^{-1}}$). $\Delta T(t)$= $T(t) - T_{\text{env}}(t)$ is the time-dependent thermal gradient between the environment and the object (°C).
Source	[2, p. 8]
Ref. By	

Number	GD4
Label	**Transformation of Experimental Data to Appropriate Function**
SI Units	–
Equation	$T(y,t) = \mathrm{fit}(Tdata, y_{TC}, dt)$ where $\mathrm{fit} : \mathbb{R}^{m\times n} \rightarrow \mathbb{R}^{n} \rightarrow \mathbb{R} \rightarrow (\mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R})$
Description	This general definition abstracts the concept of taking experimental data ($Tdata$) at the thermocouples (shown in Figure 3) over time and determining a function $T(y,t)$ that can be used in the instance models. In determining $T(y,t)$, it is assumed that the thermal resistance of the thermocouples can be ignored (A10). The specifics of the transformation of the data to a function are left as part of the design of the numerical algorithm. The options include interpolation, regression, and using the data points directly. In the instance models, it is assumed that partial derivatives of $T(y,t)$ exist and can be calculated. When needed, these partial derivatives may be calculated directly from the experimental data, or by first finding $T(y,t)$ and applying mathematical operators to it. The symbols used in the equation for this general definition are defined as follows: $T(y,t)$ is a function that takes the position, as measured from the bottom of the cylinder, and the time and returns the temperature (°C). It represents the cooling curve over time t for all locations y. y is the distance from the bottom of the cylinder (m). t is the time from the start of data collection (second). fit() is a function that takes the thermocouple data $Tdata$, the locations of the thermocouples y_{TC} and the time step dt, and returns the appropriate function $T(y,t)$. $Tdata$ is a 2D array of temperature readings. The columns correspond to each of the n different thermocouples, starting from the bottom and going up. The m rows correspond to the time of measurement. The start time for measurement is assumed to be 0 and the time between data points is assumed to be dt (A13). y_{TC} is a 1D array of position values (in m) for the n thermocouples. dt is the time between experimental measurements of the time (second). m is the number of instants of time where the thermocouple data is measured. n is the number of thermocouples.
Source	–
Ref. By	DD6, DD7, IM1

5.2.4 Data Definitions

This section collects and defines all the data needed to build the instance model. The dimension of each quantity is also given.

Number	DD1
Label	**Approximate Volumetric Heat capacity**
Symbol	$C_v(f_s)$
SI Units	$\mathrm{J\,m^{-3}\,{}^{\circ}C^{-1}}$
Equation	$C_v(f_s) = C_v^b(1 - f_s) + C_v^e f_s$
Description	$C_v(f_s)$ is the volumetric heat capacity ($\mathrm{J\,m^{-3}\,{}^{\circ}C^{-1}}$) where $C_v = \rho C_p$. C_v^b is the volumetric heat capacity at start of solidification ($\mathrm{J\,m^{-3}\,{}^{\circ}C^{-1}}$). $C_v^b = C_v^L(T_L)$, where T_L is the temperature at the liquidus point (DD6) and $C_v^L(T)$ is a function that maps a temperature T in the liquidus zone to a value of C_v. C_v^e is the volumetric heat capacity at the end of solidification ($\mathrm{J\,m^{-3}\,{}^{\circ}C^{-1}}$). $C_v^b = C_v^S(T_S)$, where T_S is the temperature at the solidus point (DD7) and $C_v^S(T)$ is a function that maps a temperature T in the solidus zone to a value of C_v. The above equation can be obtained from GD2. We assume that $C_v(f_s)$ can be expressed as a linear combination of the values at the beginning and end of the solidification (from A4).
Sources	[,]
Ref. By	IM4

Number	DD2
Label	**Approximate Thermal Diffusivity**
Symbol	$\alpha(f_s)$
SI Units	$\mathrm{m}^2\,\mathrm{s}^{-1}$
Equation	$\alpha(f_s) = \alpha_b(1 - f_s) + \alpha_e f_s$
Description	$\alpha(f_s)$ is the thermal diffusivity ($\mathrm{m}^2\,\mathrm{s}^{-1}$). $\alpha = \frac{k}{\rho C_p} = k/C_v$ where k is the thermal conductivity, ρ is the density, C_p is the specific heat capacity and C_v is the volumetric heat capacity. α_b is the thermal diffusivity at the start of the solidification ($\mathrm{m}^2\,\mathrm{s}^{-1}$). $\alpha_b = \alpha_L(T_L)$, where T_L is the temperature at the liquidus point (DD6) and $\alpha_L(T)$ is a function that maps a temperature T in the liquidus zone to a value of α. α_e is the thermal diffusivity at the end of the solidification ($\mathrm{m}^2\,\mathrm{s}^{-1}$). $\alpha_e = \alpha_S(T_S)$, where T_S is the temperature at the solidus point (DD7) and $\alpha_S(T)$ is a function that maps a temperature T in the solidus zone to a value of α. T is the temperature (°C). This can be obtained from GD2. we assume that $\alpha(f_s)$ can be expressed as a linear combination of the values at the beginning and at the end of the solidification (from A5).
Sources	[,]
Ref. By	IM4

Number	DD3
Label	**Approximate Density**
Symbol	$\rho(f_s)$
SI Units	$\mathrm{kg\,m^{-3}}$
Equation	$\rho(f_s) = \rho_b(1 - f_s) + \rho_e f_s$
Description	$\rho(f_s)$ is the density ($\mathrm{kg\,m^{-3}}$). $\rho = m/V$ where m and V are the mass and volume of the material currently under consideration. ρ_b is the density at the start of the solidification ($\mathrm{kg\,m^{-3}}$). $\rho_b = \rho_L(T_L)$, where T_L is the temperature at the liquidus point (DD6) and $\rho_L(T)$ is a function that maps a temperature T in the liquidus zone to a value of ρ. ρ_e is the density at the end of the solidification ($\mathrm{kg\,m^{-3}}$). $\rho_e = \rho_S(T_S)$, where T_S is the temperature at the solidus point (DD7) and $\rho_S(T)$ is a function that maps a temperature T in the solidus zone to a value of ρ. T is the temperature (°C). This can be obtained from GD2. We assume that $\rho(f_s)$ can be expressed as a linear combination of the values at the beginning and at the end of the solidification (from A6).
Sources	[,]
Ref. By	IM4

Number	DD4
Label	**Latent heat of solidification**
Symbol	L
SI Units	J/kg
Equation	$L = \frac{Q_s}{m}$
Description	Q_s is the amount of energy released (in J) during the phase change from liquid to solid, m is the mass of the substance (in kg), and L is the specific latent heat for a particular substance ($\mathrm{J\,kg^{-1}}$).
Source	http://en.wikipedia.org/wiki/Latent_heat
Ref. By	IM4, DD5

Number	DD5
Label	**Solid Fraction**
Symbol	f_s
SI Units	unitless
Equation	$f_s = \frac{1}{L\rho}\int_{t_b}^{t} q_s(\tau)d\tau$
Description	f_s is the solid fraction (unitless). L is the latent heat of solidification ($\mathrm{J\,kg^{-1}}$) (from DD4). t_b is the beginning of solidification in (s). t is the current time in (s). τ is the dummy time for integration in (s). ρ is the density of the metal alloy ($\mathrm{kg\,m^{-3}}$). q_s is the rate of heat generated by solidification ($\mathrm{W\,m^{-3}}$). Following Fourier's method, the fraction of solid corresponds to the ratio of the heat released up to time t to the total heat released for complete solidification. For Fourier's method to apply, it is necessary that the only mode of heat transfer is via conduction (A3). The denominator is found by rearranging GD4 to find $Q_s = Lm$, where m is the mass of the material under consideration. The numerator (heat released up to time t) is found using T2 with $\frac{dQ}{dt} = \int_V g(t)dV = g(t)V$, where g is the volumetric heat generated from T1 and V is the volume of material under consideration. To clarify that the heat generated is from solidification, g is relabelled as q_S. To use T2, the integration limit a is t_b and b is the current time t. Combining the above, $f_s = \frac{Q_s}{Lm} = \frac{V\int_{t_b}^{t} q_s(\tau)d\tau}{Lm} = \frac{V}{Lm}\int_{t_b}^{t} q_s(\tau)d\tau = \frac{1}{L\rho}\int_{t_b}^{t} q_s(\tau)d\tau$, where $\rho = m/V$.
Sources	[7, p. 113]
Ref. By	IM4

Number	DD6
Label	**Liquidus point**
Symbol	(t_L, T_L)
SI Units	°C for T_L and s for t_L
Equation	$(t_L, T_L) = \text{liquidus}(T_{y^*}(t))$, where liquidus : $(\mathbb{R} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \times \mathbb{R}$ and $T_{y^*} = T(y^*, t)$ using the function $T(y, t)$ from GD4.
Description	The liquidus temperature specifies the temperature above which a material is completely liquid, and the maximum temperature at which crystals can co-exist with the melt in thermodynamic equilibrium. The liquidus point for a given height in the cylinder (y^*) is the time (t_L) and the temperature (T_L) when the first solids start to appear. This point is detected in the cooling curve as the first point where the slope changes dramatically (as shown on Figure 1). The height (y^*) where the liquidus point is calculated will usually correspond to the location of a thermocouple, but in the interest of generality this is not required. As stated in GD4, the temperature function $T(y, t)$ is an abstraction. The calculations will likely come from the discretized experimental data directly. The symbols used in the equation for this definition are defined as follows: t_L is the time for the cooling curve when solids first appear (s). T_L is the temperature in the cooling curve when solids first appear (°C). $T(y, t)$ is the cooling curve over time t at all possible locations y (°C). (GD4) $T_{y^*}(t)$ is the cooling curve at location y^* (°C). y is the distance from the bottom of the cylinder (m). y^* is the specific distance from the bottom of the cylinder for which the liquidus point is being calculated (m). t is the time from the start of data collection (s). liquidus() is a function that takes the cooling curve $T_{y^*}(t)$ at location y^* and returns the tuple representing the liquidus point.
Sources	https://en.wikipedia.org/wiki/Liquidus
Ref. By	DD1, DD2, DD3, IM2, IM4, R12

Number	DD7
Label	**Solidus Point**
Symbol	(t_S, T_S)
SI Units	°C for T_S and s for t_S
Equation	$(t_S, T_S) = \text{solidus}(T_{y^*}(t))$, where solidus : $(\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \times \mathbb{R}$ and $T_{y^*} = T(y^*, t)$ using the function $T(y, t)$ from GD4.
Description	The solidus temperature specifies the temperature below which a material is completely solid. In other words, solidus defines the temperature at which a substance begins to melt. The solidus point for a given height in the cylinder (y^*) is the time (t_S) and the temperature (T_S) when the material is completely solid. This point is detected in the cooling curve as the end of the region of approximately zero slope (as shown on Figure 1). The height (y^*) where the solidus point is calculated will usually correspond to the location of a thermocouple, but in the interest of generality this is not required. As stated in GD4, the temperature function $T(y, t)$ is an abstraction. The calculations will likely come from the discretized experimental data directly. The symbols used in the equation for this definition are defined as follows: t_S is the time for the cooling curve when the material is first completely solid (s). T_S is the temperature in the cooling curve when the material is first completely solid (°C). $T(y, t)$ is the cooling curve over time t at all possible locations y (°C). (GD4) $T_{y^*}(t)$ is the cooling curve at location y^* (°C). y is the distance from the bottom of the cylinder (m). y^* is the specific distance from the bottom of the cylinder for which the solidus point is being calculated (m). t is the time from the start of data collection (s). solidus() is a function that takes the cooling curve $T_{y^*}(t)$ at location y^* and returns the tuple representing the solidus point.
Sources	http://link.springer.com/referenceworkentry/10.1007%2F978-3-642-11274-4_1467
Ref. By	DD1, DD2, DD3, IM1, IM3, IM4, R13

Number	DD8
Label	**Heat flux at bottom of cylinder**
Symbol	q
SI Units	$\mathrm{W\,m^{-2}}$
Equation	$q(t) = h(T_0(t) - T_{\text{env}}(t))$, over area A
Description	$T_0(t)$ is the temperature at the bottom of the cylinder over time. T_{env} is the temperature of the air around the bottom of copper plate on the bottom of the mold. The heat flux out of the plate, q, is found by assuming that Newton's Law of Cooling applies (A8). This law (GD3) is used on the surface of the copper plate, which has area A and heat transfer coefficient h ($\mathrm{W/(m^2\,°C)}$).
Sources	–
Ref. By	IM1, IM2, IM3

5.2.5 Instance Model

This section transforms the problem defined in the Section 5.1 into one which is expressed in mathematical terms. It uses concrete symbols defined in Section 5.2.4 to replace the abstract symbols in the models identified in the Sections 5.2.2 and 5.2.3.

Number	IM1
Label	**Solve Inverse Heat Transfer Problem for Heat Transfer Coefficient**
Input	$T(y,t)$ (see DD4), from which $\frac{\partial T}{\partial t}$ and $\frac{\partial^2 T}{\partial y^2}$ can be derived, as required. Material property $k_S(T)$, $C_p^S(T)$, $\rho_S(T)$ T_{env}
Output	$h(t)$ such that the following PDE and boundary conditions are satisfied using the experimental data represented in $T(y,t)$: $$\frac{\partial T}{\partial t} = \alpha_S(T)\frac{\partial^2 T}{\partial y^2}, \text{ where } \alpha_S(T) = \frac{k_S(T)}{\rho_S(T)C_p^S(T)} \qquad (1)$$ subject to the boundary conditions $q = 0$ on all boundaries, except for the bottom of the cylinder where $q(t) = h(t)(T_0(t) - T_{\text{env}}(t))$ from DD8.
Description	To determine the heat transfer coefficient at the bottom of the cylinder requires a calibration run with an alloy with known material properties. Finding the heat transfer coefficient involves solving an inverse heat transfer problem. The calculations are done on the data after solidification of the cylinder. The solidus point can be found using DD7. The governing PDE comes from GD1, but with the source term (q_s) zero, since the metal is assumed to be solid. The symbols used in this model are as follows: $T(y,t)$ is the temperature in °C found using the temperature values at the known thermocouple locations in (m) at time t (s) (from GD4) $k_S(T)$ is the thermal conductivity of the solid metal as a function of temperature ($\mathrm{W\,m^{-1}\,°C^{-1}}$) $\rho_S(T)$ is the density of the solid metal, potentially as a function of temperature ($\mathrm{kg\,m^{-3}}$) (DD3) C_p^S is the specific heat capacity of the solid metal, possibly as a function of temperature ($\mathrm{J\,kg^{-1}\,°C^{-1}}$) $\alpha_S(T)$ is the thermal diffusivity of the solid metal as a function of temperature ($\mathrm{m^2/s}$) T_{env} is the temperature of the air at the bottom of the cylinder (°C)
Sources	Some related information is available at: http://web.cecs.pdx.edu/~gerry/class/ME448/notes/pdf/
Ref. By	IM2, IM3, R7, R10, R12, R13

Number	IM2
Label	**Find α_b the thermal diffusivity at the liquidus point**
Input	$T(y,t)$ (see DD4), from which $\frac{\partial T}{\partial t}$ and $\frac{\partial^2 T}{\partial y^2}$ can be derived, as required. Heat transfer coefficient $h(t)$ using IM1 and T_{env} (t_L, T_L) from DD6
Output	α_b such that the following PDE and boundary conditions are satisfied using the experimental data represented in $T(y,t)$: $$\frac{\partial T}{\partial t} = \alpha_b \frac{\partial^2 T}{\partial y^2} \qquad (2)$$ subject to the boundary conditions $q = 0$ on all boundaries, except for the bottom of the cylinder where $q(t) = h(t)(T_0(t) - T_{\text{env}}(t))$ from DD8.
Description	To determine the liquidus thermal conductivity an inverse heat transfer problem is solved. The data used is for time 0 to time t_L. The governing PDE comes from GD1, but with the source term (q_s) zero, since the metal is assumed to be liquid. The symbols used in this model are as follows: $T(y,t)$ is the temperature in °C found using the temperature values at the known thermocouple locations in (m) at time t (s) (from GD4) α_b is the thermal diffusivity of the metal at the liquidus point ($\text{m}^2\,\text{s}^{-1}$) h heat transfer coefficient ($\text{W}/(\text{m}^2\,{}^\circ\text{C})$) T_{env} temperature of the air at the bottom of the cylinder (°C) t_L is the time for the cooling curve when solids first appear (s). T_L is the temperature in the cooling curve when solids first appear (°C)
Sources	–
Ref. By	IM4, R12, R17

Number	IM3
Label	**Find α_e the thermal diffusivity at the solidus point**
Input	$T(y,t)$ (see DD4), from which $\frac{\partial T}{\partial t}$ and $\frac{\partial^2 T}{\partial y^2}$ can be derived, as required. Heat transfer coefficient $h(t)$ using IM1 and T_{env} (t_S, T_S) from DD7
Output	α_e such that the following PDE and boundary conditions are satisfied using the experimental data represented in $T(y,t)$: $$\frac{\partial T}{\partial t} = \alpha_e \frac{\partial^2 T}{\partial y^2} \qquad (3)$$ subject to the boundary conditions $q = 0$ on all boundaries, except for the bottom of the cylinder where $q(t) = h(t)(T_0(t) - T_{\text{env}}(t))$ from DD8.
Description	To determine the solidus thermal conductivity an inverse heat transfer problem is solved. The data used is for time t_S to the end of the experiment. The governing PDE comes from GD1, but with the source term (q_s) zero, since the metal is assumed to be solid. The symbols used in this model are as follows: $T(y,t)$ is the temperature in °C found using the temperature values at the known thermocouple locations in (m) at time t (from GD4) (s) α_e is the thermal diffusivity of the solid alloy ($\text{m}^2\,\text{s}^{-1}$) h heat transfer coefficient ($\text{W}/(\text{m}^2\,{}^\circ\text{C})$) T_{env} temperature of the air at the bottom of the cylinder (°C) t_S is the time for the cooling curve when the material has first solidified (s). T_S is the temperature in the cooling curve when it first solidifies (°C)
Sources	–
Ref. By	IM4, R13, R18

Number	IM4
Label	**Find Fraction Solid f_s as a Function of Temperature**
Input	$T(y,t)$ (see DD4), from which $\frac{\partial T}{\partial t}$ and $\frac{\partial^2 T}{\partial y^2}$ can be derived, as required Material properties $C_v^L(T)$, $C_v^S(T)$, $\rho_L(T)$, $\rho_S(T)$, α_b (from IM2), α_e (from IM3), and L y^*, (t_L, T_L) from DD6 and (t_S, T_S) from DD7
Output	Solve $f_s(t)$ at location y^* such that the following ODE is satisfied with $f_s(t_L) = 0$: $\dot{f}_s(f_s, t) = \frac{C_v(f_s)}{L\rho(f_s)}\left[\frac{\partial T(t)}{\partial t} - \alpha(f_s)\frac{\partial^2 T(t)}{\partial y^2}\right]$ where $C_v(f_s) = C_v^b(1 - f_s) + C_v^e f_s$ $\rho(f_s) = \rho_b(1 - f_s) + \rho_e f_s$ $\alpha(f_s) = \alpha_b(1 - f_s) + \alpha_e f_s$ where $C_v^b = C_v^L(T_L)$, $C_v^e = C_v^S(T_S)$, $\rho_b = \rho_L(T_L)$, $\rho_e = \rho_S(T_S)$. After solving the ODE, $f(t)$ and $T(t)$ are known, which means that $f(T)$ can be shown.
Description	$T(y,t)$ is the temperature readings in (°C) found by the temperature values at the known thermocouple locations in (m) at time t (s) $C_v^L(T)$, $C_v^S(T)$ is the volumetric heat capacity of the alloy as a function of temperature, as a liquid and a solid, respectively (J/(m^3 °C)) $\rho_L(T)$, $\rho_S(T)$ is the density as a function of temperature for the liquid and solid alloy, respectively (kg/m^3) α_b, α_e is the thermal diffusivity at the liquidus point and solidus point, respectively (m^2/s) (IM2) L is the specific latent heat for a particular substance (J kg^{-1}) $C_v(f_s)$ is the volumetric heat capacity in the two phase zone (J/(m^3 °C)) (DD1) $\alpha(f_s)$ is the thermal diffusivity in the two phase zone (m^2/s) (DD2) $\rho(f_s)$ is the density of the alloyin the two phase zone (kg m^{-3}) (DD3) y^* is the specific distance from the bottom of the cylinder for which the solid fraction is being calculated (m). The detailed derivation is below.
Sources	–
Ref. By	R14, R19

Derivation of the Fraction Solid

To find the rate of change of f_s with respect to temperature, we start from the conservation of thermal energy equation. From GD1 we have:

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial y^2} + \frac{q_s}{C_v} \tag{4}$$

To find q_s in terms of f_s, we differentiate DD5 and rearrange to obtain:

$$q_s = \dot{f}_s L \rho \tag{5}$$

Here density ρ is assumed constant based on our assumption A12. The above equation for q_s also appears in [5].

Substituting Equation 5 into Equation 4 we obtain

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial y^2} + \frac{\dot{f}_s L \rho}{C_v} \tag{6}$$

Rearranging the above, we obtain:

$$\dot{f}_s = \frac{C_v}{L\rho} \left[\frac{\partial T}{\partial t} - \alpha \frac{\partial^2 T}{\partial y^2} \right] \tag{7}$$

To make the functional dependence explicit, Equation 5.2.5 can be written as:

$$\dot{f}_s(f_s, t) = \frac{C_v(f_s)}{L\rho(f_s)} \left[\frac{\partial T(t)}{\partial t} - \alpha(f_s) \frac{\partial^2 T(t)}{\partial y^2} \right] \tag{8}$$

For the initial conditions, we know that $f_s(t_L) = 0$, since at the liquidus temperature solids just start to form. We also know that we should get the result that $f_s(t_S) = 1$.

5.2.6 Data Constraints

Table 1 shows the data constraints on the input variables for the configuration mode. The configuration mode is where the data that is typically stable across multiple runs is entered. The relevant data consists of the height of the cylindrical mold (H), the diameter of the mold (D), the number of thermocouples (n) and their locations (y_{TC}).

The column for physical constraints gives the physical limitations on the range of values that can be taken by the variables. The column for software constraints restricts the range of inputs to reasonable values. The constraints are conservative, to give the user of the model the flexibility to experiment with unusual situations. The column of typical values is intended to provide a feel for a common scenario. The uncertainty column (labelled UC) provides an estimate of the confidence with which the physical quantities can be measured. This information would be part of the input if one were performing an uncertainty quantification exercise.

Table 1: Input Variables for Configuration Mode

Var[a]	Type [b]	Physical Constraints	Software Constraints	Typical Value	UC
H[c]	$\mathbb{R}$	$H > 0$	$H_{\text{min}} \leq H \leq H_{\text{max}}$	0.15 m	10%
D[d]	$\mathbb{R}$	$D > 0$	$D_{\text{min}} \leq D \leq D_{\text{max}}$	0.05 m	10%
n	$\mathbb{N}$	$n > 0$	$n \leq n_{\text{max}}$	8	NA
y_{TC}	$\mathbb{R}^n$	$\forall(i : \mathbb{N} \mid i \in [1..n] \cdot 0 \leq y_{TCi} \leq H)$	$\forall(i : \mathbb{N} \mid i \in [1..n-1] \cdot y_{TCi+1} > y_{TCi})$	y_{TC}[e] m	10%
dt	$\mathbb{R}$	$dt > 0$	$dt_{\text{min}} \leq dt \leq dt_{\text{max}}$	0.01 s	10%

[a] The symbols used in this table are explained in GD4

[b] Type refers to the data type of the variable

[c] Height of the cylinder

[d] Diameter of the cylinder

[e] $y_{TC} = [0.005, 0.015, 0.025, 0.040, 0.060, 0.080, 0.1, 0.13]$

Table 2 summarizes the input variables for transforming the m experimental temperature readings for each of the n thermocouples ($Tdata$) to the function $T(y,t)$. The symbols used in this table are explained in GD4.

Table 2: Input Variables for Calculating $T(y,t)$

Var[a]	Type	Physical Constraints	Software Constraints	Typical Value	UC
m	$\mathbb{N}$	$m > 0$	$m \leq m_{\text{max}}$	7000	NA
$Tdata_{ij}$[b]	$\mathbb{R}$	$Tdata_{ij} > \text{AbsZero}$	$T_{\text{min}} \leq Tdata_{ij} \leq T_{\text{max}}$	600 °C	10%

[a] The symbols used in this table are explained in GD4

[b] $i \in [1..m], j \in [1..n]$

Both the calibration and calculation modes require material properties for the alloys that are being cooled. Table 3 summarizes the data constraints on the material properties. In addition, the table includes the operating conditions as given by the value of the temperature of the environment around the bottom of the cylinder T_{env}.

Tables 1, 2 and 3 are parameterized by various constants. The values of these constants are given in the specification parameters table in the Appendix (Table 6).

Table 3: Input Variables for Material Properties and Operating Conditions

Var	Type	Physical Constraints	Software Constraints	Typical Value	UC
k_S	$\mathbb{R} \rightarrow \mathbb{R}$	$\forall(T : \mathbb{R} \vert \cdot k_S(T) > 0)$	$\forall(T : \mathbb{R} \vert \cdot k_{S\text{min}} \leq k_S(T) \leq k_{S\text{max}})$	180 W/(m °C)	10%
ρ_S	$\mathbb{R} \rightarrow \mathbb{R}$	$\forall(T : \mathbb{R} \vert \cdot \rho_S(T) > 0)$	$\forall(T : \mathbb{R} \vert \cdot \rho_{S\text{min}} \leq \rho_S(T) \leq \rho_{S\text{max}})$	2700 kg/m^3	10%
C_P^S	$\mathbb{R} \rightarrow \mathbb{R}$	$\forall(T : \mathbb{R} \vert \cdot C_P^S(T) > 0)$	$\forall(T : \mathbb{R} \vert \cdot C^S_{P\text{min}} \leq C_P^S(T) \leq C^S_{P\text{max}})$	1 J/(kg °C)	10%
T_{env}	$\mathbb{R}$	$T_{\text{env}} > \text{AbsZero}$	$T_{\text{env}\text{min}} \leq T_{\text{env}} \leq T_{\text{env}\text{max}}$	25 °C	10%
C_v^L	$\mathbb{R} \rightarrow \mathbb{R}$	$\forall(T : \mathbb{R} \vert \cdot C_v^L(T) > 0)$	$\forall(T : \mathbb{R} \vert \cdot C^L_v\text{min} \leq C_v^L(T) \leq C^L_v\text{max})$	2700 J/(m^3 °C)	10%
C_v^S	$\mathbb{R} \rightarrow \mathbb{R}$	$\forall(T : \mathbb{R} \vert \cdot C_v^S(T) > 0)$	$\forall(T : \mathbb{R} \vert \cdot C^S_v\text{min} \leq C_v^S(T) \leq C^S_v\text{max})$	2700 J/(m^3 °C)	10%
ρ_L	$\mathbb{R} \rightarrow \mathbb{R}$	$\forall(T : \mathbb{R} \vert \cdot \rho_L(T) > 0)$	$\forall(T : \mathbb{R} \vert \cdot \rho_{L\text{min}} \leq \rho_L(T) \leq \rho_{L\text{max}})$	2700 kg/m^3	10%
L	$\mathbb{R}$	$L > 0$	$L_{\text{min}} \leq L \leq L_{\text{max}}$	397500 J/kg	10%

Table 4: Place holder for Unidirectional constraint

Var	Physical Constraints (for unidirectional constraint)
var	*constraint*

5.2.7 Properties of a Correct Solution

In addition to the properties that must be true of the input variables, constraints also exist on valid outputs. The properties that need to be true of correct solutions are summarized in Table 5. This table includes entries for the output of the calibration mode and the calculation mode. In the calibration mode (IM1) the output is the heat transfer coefficient as a function of time, $h(t)$. In the calculation mode (IM2, IM3 and IM4), the outputs are the thermal diffusivity and the beginning and end of solidification (α_b and α_e) and the solid fraction as a function of temperature ($f_s(T)$).

Table 5: Output Variables

Var	Physical Constraints
h	$h > 0$
α_b	$k_b > 0$
α_e	$k_e > 0$
f_s	$0 \leq f_s \leq 1$

6 Requirements

This section provides the functional requirements, the business tasks that the software is expected to complete, and the nonfunctional requirements, the qualities that the software is expected to exhibit.

6.1 Functional Requirements

The functional requirements are split between three different modes: configuration mode, calibration mode and f_s calculation mode.

6.1.1 Configuration Mode

The experimental set-up is not expected to vary too often. Therefore, the associated data is considered separately as configuration information. The important quantities to enter are the dimensions of the cylindrical mold, the locations of the thermocouples and the sampling rate.

R1: Input the following quantities, which define the dimensions of the cylindrical mold, the locations of the thermocouples and the sampling rate:

symbol	**type**	**unit**	**description**
H	$\mathbb{R}$	m	height of cylindrical mold
D	$\mathbb{R}$	m	diameter of cylindrical mold
n	$\mathbb{N}$	NA	number of thermocouples
y_{TC}	$\mathbb{R}^n$	m	locations of n thermocouples
dt	$\mathbb{R}$	s	time step between collected data points

R2: Input the symbolic parameters (used for input verification) specified in the Appendix in Table 6.

R3: Verify that the dimensions of the cylinder are consistent with the assumption of unidirectional heat flow (A2). (*Dr. Mohamed Hamed mentioned perviously that he had a means to judge the validity of the assumption, but this information has not yet been collected*). If the dimensions do not satisfy the unidirectional heat flow condition, then a warning message will be issued.

R4: Verify that the configuration input data satisfies the constraints listed in Table 1.

R5: The configuration data should be persistent between program runs, unless it is explicitly changed.

6.1.2 Calibration Mode

In the calibration mode an experiment is run with an alloy where the material properties are known. This allows for calculation of the heat transfer coefficient at the base of the cylinder.

R6: After running the calibration experiment, input the temperature data collected from the thermocouples: $Tdata$, n (GD4).

R7: Input the necessary material properties for the known alloy and the temperature of the environment: $k_S(T)$, $C_p^S(T)$, $\rho_S(T)$ and T_{env} (IM1).

R8: Verify the temperature data against the constraints in Table 2.

R9: Verify the material properties and operating conditions against the constraints in Table 3.

R10: Using the input information (R7 and R7) and configuration information (R1), calculate the heat transfer coefficient ($h(t)$) (IM1).

6.1.3 Calculation Mode

Once the heat transfer coefficient for the experimental set-up is known, the calculations can move to finding the solid fraction as a function of temperature. This involves first finding the material properties at the liquidus and solidus points, and then using this information to solve for the evolution of the fraction solid in the two phase zone.

R11: After running the experiment, input the temperature data collected from the thermocouples: $Tdata$, n (GD4).

R12: Input the necessary data to calculate α_b, which includes the following:

- Heat transfer coefficient and environment temperature: $h(t)$ and T_{env} (IM1)
- Liquidus point: (t_L, T_L) (DD6)

R13: Input the necessary data to calculate α_e, which includes the following:

- Heat transfer coefficient and environment temperature: $h(t)$ and T_{env} (IM1)
- Solidus point: (t_S, T_S) (DD7)

R14: Input the additional information necessary to solve for f_s: L (IM4)

R15: Verify the temperature data against the constraints in Table 2.

R16: Verify the material properties and operating conditions against the constraints in Table 3.

R17: Using the input information (R12) and configuration information (R1), calculate the value of α_b (IM2).

R18: Using the input information (R13) and configuration information (R1), calculate the value of α_e (IM3).

R19: Using the temperature data (R11) and other input information (R12, R13 and R14) and configuration information (R1), calculate the value of $f_s(t)$ (IM4).

R20: Using $f_s(t)$ and $T(t)$ find $f_s(T)$ (IM4).

6.2 Non-Functional Requirements

throughout

6.2.1 Look and Feel Requirements

- 85% of surveyed users shall be able to see button labels and text descriptions on the screen.
- 85% of surveyed users shall agree that the software provides an adequate amount of data visualization for the user.

6.2.2 Usability and Humanity Requirements

- 85% of surveyed users shall agree that the software is simple enough to learn and use.
- Both Fahrenheit and Celsius values shall be available.
- A single workflow shall be applicable for different alloys.

6.2.3 Installability Requirements

- 90% of surveyed users shall agree that the software has a simple installation process, comparable to a typical application.

6.2.4 Performance Requirements

- Software startup time shall be equal or better than other comparable applications on all operating systems.
- 90% of surveyed users shall agree that the response times of non-calculation actions of the user interface are better or equal to other comparable software.
- Software response time for calculation actions shall have a response time of less than 15 minutes.

- A database query shall take less than 1 minute.
- All calculation results shall have the correct amount significant digits according to the input values.
- Predicted values of a temperature at a certain time and location shall be within a 2% deviation from the test values of a closely related location at the same given time.
- The system shall always have a coefficient value of cooling available even if the database the system interfaces with is down/offline.

6.2.5 Operating and Environmental Requirements

- The software shall be able to interface with Hayley's database system.
- The software shall interact with the building's internet network which is connected to the World Wide Web to use a remote database.
- The software shall be able to run on Windows 7/10, Linux, and Mac OSX operating systems.

6.2.6 Maintainability and Support Requirements

- Additional features or bug fixes for the software shall take less than 10% of software development resources.
- The software shall have the ability to log software errors.

6.2.7 Security Requirements

- Not applicable.

6.2.8 Cultural Requirements

- Not applicable.

6.2.9 Compliance Requirements

- Not applicable.

7 Likely Changes

LC1: The heat transfer coefficient at the bottom of the cylinder could potentially depend on temperature (A9).

LC2: The thermocouple data is currently assumed to start at time 0 and have a constant time step (A13). However, a more complex form of the thermocouple data is possible.

8 Unlikely Changes

The changes listed here are considered to be so significant that they would fundamentally change SFS. If these changes become necessary, significant changes will be necessary for the SRS and subsequent documents.

UC1: If for some reason forms of energy other than thermal energy (A1) had to be considered, the software will be invalid.

UC2: If heat flow is not unidirectional A2 the software will be invalid.

References

[1] Attila Diószegi and Jesper Hattel. Inverse thermal analysis method to study solidification in cast iron. *International Journal of Cast Metals Research*, 17(5):311–318, 2004.

[2] F. P. Incropera, D. P. Dewitt, T. L. Bergman, and A. S. Lavine. *Fundamentals of Heat and Mass Transfer*. John Wiley and Sons, United States, sixth edition edition, 2007.

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

[4] 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.

[5] Hacı Mehmet Şahin, Kadir Kocatepe, Ramazan Kayıkcı, and Neşet Akar. Determination of unidirectional heat transfer coefficient during unsteady-state solidification at metal casting–chill interface. *Energy conversion and management*, 47(1):19–34, 2006.

[6] 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*, pages 107–121, Paris, France, 2005. In conjunction with 13th IEEE International Requirements Engineering Conference.

[7] Doru Stefanescu. *Science and Engineering of Casting Solidification*. Kluwer Academic/Plenum Publishers, 2002.

A Supporting Information

Table 6: Specification Parameter Values

Var	**Value**
AbsZero	273.15 °C
H_{min}	0.001 m
H_{max}	100 m
D_{min}	0.001 m
D_{max}	100 m
m_{max}	1000
dt_{min}	0.0001 s
dt_{max}	1000 s
n_{max}	1×10^4
T_{min}	100 °C
T_{max}	1×10^4 °C
$k_{L\text{min}}$	0.001 W/(m °C)
$k_{L\text{max}}$	1×10^5 W/(m °C)
$k_{S\text{min}}$	0.001 W/(m °C)
$k_{S\text{max}}$	1×10^5 W/(m °C)
$\rho_{S\text{min}}$	0.001 kg m^{-3}
$\rho_{S\text{max}}$	1×10^4 kg m^{-3}
$\rho_{L\text{min}}$	0.001 kg m^{-3}
$\rho_{L\text{max}}$	1×10^4 kg m^{-3}
$C_{P\text{min}}^{L}$	1×10^{-4} $\text{J kg}^{-1}\,°\text{C}^{-1}$
$C_{P\text{max}}^{L}$	1×10^{4} $\text{J kg}^{-1}\,°\text{C}^{-1}$
$C_{P\text{min}}^{S}$	1×10^{-4} $\text{J kg}^{-1}\,°\text{C}^{-1}$
$C_{P\text{max}}^{S}$	1×10^{4} $\text{J kg}^{-1}\,°\text{C}^{-1}$
$C_{v}^{L}\text{min}$	1×10^{-4} $\text{J kg}^{-1}\,°\text{C}^{-1}$
$C_{v}^{L}\text{max}$	1×10^{4} $\text{J kg}^{-1}\,°\text{C}^{-1}$
$C_{v}^{S}\text{min}$	1×10^{-4} $\text{J kg}^{-1}\,°\text{C}^{-1}$
$C_{v}^{S}\text{max}$	1×10^{4} $\text{J kg}^{-1}\,°\text{C}^{-1}$
L_{min}	0.001 m
L_{max}	1×10^{10} m
$T_{\text{env}_{\text{min}}}$	0 °C
$T_{\text{env}\text{max}}$	100 °C

Appendix C

Task based inspection: Task List for Scientists - SRS review

Task list for Scientists - SRS Review

Malavika Srinivasan and Spencer Smith

December 3, 2018

Contents

1 Purpose of Document

This document is intended to act as a guide to review the SRS document. The scope of this document is to involve the scientists in reading and reviewing the SRS document. To initiate the review process, we have assigned a set of tasks which needs to be completed. Every task is framed as a question in a specific section of the SRS. Each question is to be answered after reading the corresponding section in the SRS document. The tasks will be assigned to the reviewers, and the responses recorded, using GitLab issue tracking.

An SRS is an abstract document which says *what* problem is being solved, but do *how* to solve it. The SRS is used as a starting point for subsequent development phases, including writing the design specification and the software verification and validation plan. Review of SRS document is important to reach a common platform between software engineers and scientists. Any changes required in the software are finalized after the review of SRS. A properly reviewed SRS acts as an agreement between the scientists and the software engineers regarding the deliverables of the project.

2 Questions for Dr. Kumar

We would like all the scientists involved in this project to go through the SRS document fully, review the document and give us suggestions. However we understand if you cannot go through the entire document. Below are the specific questions that we would like to clarify with you. Please respond on the GitLab issue tracker.

1: Please go through the System Context and let us know if it is complete and unambiguous. - **Section 4.1 in SRS**

2: Please read assumption A2 and let us know if this is acceptable with respect to SFS. Are you aware of a mathematical bounds on the cylinder dimension or aspect ratio that define the limits of applicability of this assumption? - **Section 5.2.1 Assumption A2 in SRS**

3: Please read assumptions A3, A4, A5 and A6. In these assumptions, we have assumed that the material properties can be expressed as a linear combination of their values at the beginning and end of solidification. Please let us know if this seems reasonable with respect to SFS. - **Section 5.2.1 Assumption A3, A4, A5 and A6 in SRS**

4: Please read assumption A9. We have assumed that the thermal resistance due to thermocouples is negligible. Please let us know if this assumption is reasonable for SFS. - **Section 5.2.1 Assumption A9 in SRS**

5: Please read assumption A12 and let us know if it is reasonable with respect to SFS. - **Section 5.2.1 Assumption A12 in SRS**

6: Please let us know if the Data Definition 1 - Solid Fraction is explained clearly. Can you also please verify if the units in LHS and RHS of the f_s expression match. For eg: f_s is unitless. So, RHS should be unitless as well. If you are aware of a good reference that explains the material covered in the definition, please let us know. - **DD1 in section 5.2.4 in SRS**

7: The Data Definitions DD2, DD3, DD4 and DD5 are based on the assumption A3, A4, A5 and A6. Please let us know if the definitions are correct and have been explained clearly. If you are aware of any reference from literature, where they have used this idea in a similar situation, please let us know and we will cite it. - **DD2, DD3, DD4 and DD5 in section 5.2.4 in SRS**

8: Please let us know if the Data Definition DD6 - *Latent Heat of Solidification* is explained clearly. Also can you please verify if the units in

LHS and RHS of the L expression match. If you are aware of a good reference that explains the material covered in the definition, please let us know. - **DD6 in section 5.2.4 in SRS**

9: Please let us know if the Data Definitions DD7, DD8 and DD9 - *Liquidus point*, *Eutectic point* and *Solidus point* are explained clearly. If you have any other theoretical definition for the same, please explain. If you are aware of a good reference that explains the material covered in the definition,please let us know. - **DD7, DD8 and DD9 in section 5.2.4 in SRS**

10: Read IM1 and let us know if it is complete, correct and unambiguous. Also for the calibration run, we assume that the metal starts to freeze shortly after being poured and the equation (1) in IM1 is based on this assumption and GD1. Is this a reasonable assumption? - **IM1 in section 5.2.5 in SRS**

11: Read IM2, IM3 and let us know if they need more explanation. q is assumed to be 0 on all boundaries other than the bottom of the cylinder. Please confirm if this is correct. Also please check if the units on both sides of the equation 2 and 3 in IM2 and IM3, respectively, match. - **IM2 and IM3 in section 5.2.5 in SRS**

12: Read IM4 and let us know if the equation for $\dot{fs}$ is correct. Does the units of LHS and RHS match for the $\dot{fs}$ equation? - **IM4 in section 5.2.5 in SRS**

13: The IM4 is followed by the derivation of the fraction solid. Please review the derivation and let us know if all the steps can be followed easily. If any of the steps are missing or unclear, please let us know. - **Derivation of IM4 in section 5.2.5 in SRS**

3 Questions for Dr. Shankar

We would like all the scientists involved in this project to go through the SRS document fully, review the document and provide us with suggestions. However we understand if you cannot review the entire document. Below are the specific questions that we wanted to clarify with you. Please respond using the GitLab issue tracker.

14: Please go through the System Context and let us know if it is complete and unambiguous. - **Section 4.1 in SRS**

15: Please read assumption A2 and let us know if this is acceptable with respect to SFS. Are you aware of a mathematical bounds on the cylinder dimension or aspect ratio that define the limits of applicability of this assumption? - **Section 5.2.1 Assumption A2 in SRS**

16: Please read assumptions A3, A4, A5 and A6. In these assumptions, we have assumed that the material properties can be expressed as a linear combination of their values at the beginning and end of solidification. Please let us know if this seems reasonable with respect to SFS. - **Section 5.2.1 Assumption A3, A4, A5 and A6 in SRS**

17: Please read assumption A9, we have assumed that the thermal resistance due to thermocouples is negligible. Please let us know if this assumption is reasonable for SFS. - **Section 5.2.1 Assumption A9 in SRS**

18: Please read assumption A12 and let us know if it is reasonable with respect to SFS. - **Section 5.2.1 Assumption A12 in SRS**

19: Can you please tell us the mathematical characterization that defines the liquidus and solidus points with respect to the data in the cooling curve? We need this information to automatically extract these points from the thermocouple data. - **DD6, DD7 in section 5.2.4 in SRS**

20: We understand that G is the temperature gradient at a point when solid-liquid interface passes through it. How should G be calculated? If possible, please point us to a reference related to G. - **DD9 in section 5.2.4 in SRS**

21: We understand that R is the Velocity of the solid-liquid interface when it passes a given location. What is the mathematical definition of R? If possible, please point us to a reference related to R. - **DD10 in section 5.2.4 in SRS**

22: Read IM1 and let us know if it is correct, complete and unambiguous. Also, for the calibration run, we assume that the metal starts to freeze shortly after poured and the equation(1) in IM1 is based on this assumption and GD1. Is this a reasonable assumption? - **IM1 in section 5.2.5 in SRS**

23: Please let us know whether the material property input for the inverse heat transfer problem in IM1 should just be α, or if it should be k, C_p and ρ? IM1 is written such that only α is needed which will be an input from the user. However, for practical purposes, we might want to have the program calculate α from the other material properties like k, C_p and ρ. - **IM1 in section 5.2.5 in SRS**

24: Read IM2, IM3 and let us know if they need further explanation. q is assumed to be 0 on all boundaries except the bottom. Please confirm if this is correct. Also please check if the units on both sides of the equation 2 and 3 in IM2 and IM3 match. - **IM2 and IM3 in section 5.2.5 in SRS**

25: Read IM4 and let us know if the equation for $\dot{fs}$ is correct. Does the units of LHS and RHS match for the $\dot{fs}$ equation? - **IM4 in section 5.2.5 in SRS**

26: The IM4 is followed by the derivation of the fraction solid. Please review the derivation and let us know if all the steps can be followed easily. If any of the steps are missing or you feel lacks continuity and need further explanation, please let us know. - **Derivation of IM4 in section 5.2.5 in SRS**

27: In IM4, we could reduce the number of material properties needed by substituting C_V with $C_P * \rho$. The values of ρ in the numerator and denominator would cancel and hence the material properties needed would be only C_P and L. Is there any value to doing this, or is it better with C_V and ρ explicitly in the equation? - **IM4 in section 5.2.5 in SRSSFS**

28: Please read Requirement R3. During the last quarterly meeting Dr. Hamed informed us that he would be able to give the aspect ratio for the cylinder so that it can be made as a constraint to use the software.

Outside the constraint, the 1D heat transfer assumption wouldn't hold. Should the input verification check the validity of the input dimensions? If so, please provide the necessary information. - **R3 in section 6.1 in SRS**

29: Please read the non functional requirements and let us know if you would like to add anything to it. - **Section 6.2 in SRS**

30: Currently we have all the inputs in a *.csv* file. Is there a possibility that the input file will be in any other format other than *.csv*?

31: Please refer to Table 3 which contains a list of inputs to SFS. Are there any other inputs to SFS other than those mentioned in the table?

32: Can you please go through the elements in the Table 5 "Specification Parameter values" and let us know if the values assumed are reasonable or require any changes. The values given in this table are mostly to define extreme bounds for the input variables. The bounds are extreme enough that exceeding them is obviously an error. - **Table 5 in SRS.**

Appendix D

Code for SFS

D.1 Temperature Module

```
## @file TData.py
# @author David Li, Malavika Srinivasan, Spencer Smith
# @brief Abstract object for storing temperature data at each thermocouple
# @date 05/05/2018
import csv
import numpy.polynomial.polynomial as poly
import numpy as np
import math
from scipy import interpolate
from Util.Exceptions import *
from Util import Load
from Modules.PiecewiseADT import PiecewiseADT
from Modules.Configuration import Configuration



## Initialize Temperature Module using a data file.
# This must be called before using any other function in this module.

def init():
    global S, Y, tL, TL, tS, TS, TCNums
    S = [] # !< sequence of PiecewiseT
    Y = [] # !< sequence of Real Numbers (Height)
    tL = [] # !< sequence of Real Numbers (Liquidus Point time)
    TL = [] # !< sequence of Real Numbers (Liquidus Point temperature)
    tS = [] # !< sequence of Real Numbers (Solidus Point time)
    TS = [] # !< sequence of Real Numbers (Solidus Point temperature
    TCNums = [] # !< sequence of thermocouple numbers


## @brief This method adds data
# @details Each thermocouple is described by a series of parameters namely -
# S - Piece wise polynomial coefficients, Y - Location, tL - Liquidus time, Tl - Liquidus temp,
# tS - Solidus time, TS - Solidus temp, tc_num - Thermocouple number. All these informations are stored together in the TData
# module.<br>
# @param s(Piecewise coefficients), y(Thermocouple location), tl(Liquidus time), Tl(Liquidus temp),
# ts(Solidus time), Ts(Solidus temp),tc_num(Thermocouple number)<br>
# This method is used by Load.py.
# @todo Link Load.py method
# @exception none
# @see Load.py
# @return none
def add(s, y, tl, Tl, ts, Ts, tc_num):
    global S, Y, tL, TL, tS, TS, TCNums

    S.append(s);
    Y.append(y);
    tL.append(tl)
    TL.append(Tl);
    tS.append(ts);
    TS.append(Ts)
    TCNums.append(tc_num);


## @brief Remove unwanted thermocouple
# @details This method removes the unwanted thermocouple by maintaining an index of unwanted thermocouples.
# If the given thermocouple number matches the number in unwanted index, all the relevant data to that thermocouple
# like S, y, Liquidus and solidus points etc are removed.<br>
# This method is called by removeThermocouple() from calibration.py which
```

```
    is called by removeTC() method from CalibrationScreen.py.
# @param i - Thermocouple number
# @exception Out of bounds - When the thermocouple index to be removed
    doesn't match the index list obtained using add().
# @return none
def rm(i):
    if i in TCNums:
        try:
            index_remove = TCNums.index(i);
            del S[index_remove], Y[index_remove], tL[index_remove], TL[
                index_remove], tS[index_remove], TS[
                index_remove], TCNums[index_remove]
        except IndexError:
            print("TData.rm(i):␣Index␣out␣of␣bounds") # raise exception
                instead
        return None


## @brief Get thermocouple data
# @details This method is used to get the coefficients of ith thermocouple.
# This is called by the getBreakPoints() method from calibration.py to obtain
     information about a thermocouple.
# @param i - Thermocouple number
# @todo Link Calibration.py
# @exception Out of bounds - Raises this exception when the index number
    in parameter doesnt match index in s[i] from piecewise module.
# @see Calibration.py
# @return none
def getC(i):
    try:
        j = 0
        print('Printing␣S[i]␣from␣getC␣TData')
        print(S[i])
        return S[i]
    except IndexError:
        print("TData.getC(i):␣Index␣out␣of␣bounds") # raise exception
    return None


## @brief Liquidus point of a thermocouple
```

```
# @details This method returns the liquidus point of a thermocouple. This
    information is loaded into this module using add() from Load.py.
# @param i - Thermocouple number
# @exception Out of bounds - Raises this exception when the index number
    in parameter doesnt match index in s[i] from piecewise module.
# @return Liquidus time, Liquidus temp
def Liq(i):
    try:
        return tL[i], TL[i]
    except IndexError:
        print("TData.Liq(i):␣Index␣out␣of␣bounds") # raise exception
    return None


## @brief Solidus point of a thermocouple
# @details This method returns the Solidus point of a thermocouple. This
    information is loaded into this module using add() from Load.py.
# @param i - Thermocouple number
# @exception Out of bounds - Raises this exception when the index number
    in parameter doesnt match index in s[i] from piecewise module.
# @return Solidus time, Solidus temp
def Sol(i):
    try:
        return tS[i], TS[i]
    except IndexError:
        print("TData.Liq(i):␣Index␣out␣of␣bounds") # raise exception
    return None


## @brief Temp across all thermocouple at any time t
# @details This method gives the temperature across all thermocouples at
    any given time t. It uses feval() from PiecewiseADT module.
# This method is used in T(y, t) and visual analysis.
# @param t - time
# @exception Value error - If no data is found for the thermocouples at time t
    .
# @see PiecewiseADT.py
# @return Location value of all thermocouple, temp at all thermocouples at
    time t.
# @note This generates a value error if we dont have temp values at any
```

```
    given time. I cannot put it in exception. Please suggest.
## Return the height array and the array of piecewise functions evaluated at
    time t.
def slice(t):
    temp = []
    y = []
    for i in range(len(S)):
        try:
            temp.append(S[i].feval(t))
            y.append(Y[i])
            # print (temp)
            # print (y)

        except ValueError:
            pass
    if (len(y) == 0):
        raise ValueError('No␣valid␣data␣points␣at␣this␣time')
    return y, temp;


## @brief Finds breakpoint values from PiecewiseADT module.
# @details This method is used by __Tinterp().
# @param none
# @exception none
# @see PiecewiseADT.py
# @return Y - Location array, bkA - first break point at all locations, bkB -
    Second breakpoint at all locations.
# @note This method has to be removed if __Tinterp() is not going to be used
    .
## Return 3 values, height array, array of first breakpoints
def breakPts():
    bkA = []
    bkB = []
    for p in S:
        bkA.append(p.get__x1())
        bkB.append(p.get__x2())
    return Y, bkA, bkB


## @brief Finds temperature
```

```
# @details This method is used to find the temperature at a given location
    and time value. Works based on cubic interpolation.
# @param y - location, t - time
# @exception type error
# @return Temperature value
# @note This again throws a type error. Not sure if this has to be mentioned
    in exception.
def T(y, t): # this works well
    try:
        yy, Ty = slice(t)
        Tint = interpolate.interp1d(yy, Ty, kind='cubic')
        return Tint(y)
    except TypeError:
        return None


## @brief Finds temperature
# @details This method is used to find the temperature at a given location
    and time value. Works based on polynomial fit.
# @param y - location, t - time
# @exception type error
# @return Temperature value
# @note This again throws a type error. Not sure if this has to be mentioned
    in exception.
def TfullInterp(y, t): # with regression - not good for dTdt, but
    # smooths out d2Tdy2
    # assumes 10 thermocouples
    try:
        yy, Ty = slice(t)
        Tz = np.polyfit(yy, Ty, 9)
        p = np.poly1d(Tz)
        return p(y)
    except TypeError:
        return None


## @brief Finds temperature
# @details This method is used to find the temperature at a given location
    and time value. Works based on spline.
# @param y - location, t - time
# @exception type error
# @return Temperature value
# @note This again throws a type error. Not sure if this has to be mentioned
   in exception.
def Treg(y, t): # with regression - not good for dTdt, but smooths out
    # d2Tdy2, probably too much smoothing
    try:
        yy, Ty = slice(t)
        Tz = interpolate.splrep(yy, Ty, k=5, s=0)
        return interpolate.splev(y, Tz, der=0)
    except TypeError:
        return None


## @brief Finds temperature
# @details This method is used to find the temperature at a given location
   and time value. Works based on cubic spline.
# @param y - location, t - time
# @exception type error
# @return Temperature value
# @note This again throws a type error. Not sure if this has to be mentioned
   in exception.
def Tspline(y, t): # answers apparently pretty much the same as cubic
   interpolation
    try:
        yy, Ty = slice(t)
        Tspline = interpolate.CubicSpline(yy, Ty)
        return Tspline(y)
    except TypeError:
        return None


## @brief Finds temperature
# @details This method is used to find the temperature at a given location
   and time value. It works based on interpolation.
# @param y - Location, t - time
# @todo Needs to be redone if this is the final approach to find temperature
   at given y,t.
# @exception none
# @return Temperature value
```

```
def _T(y, t):
    yy, Ty = slice(t)
    yintPts1, TintPts1, yintPts2, TintPts2, yintPts3, TintPts3 = _Tinterp(y,
        t)

    if (yintPts1[0] <= y <= yintPts1[-1]):
        if (len(yintPts1) == 1):
            return TintPts1[0]
        else:
            Tintfun = interpolate.interp1d(yintPts1, TintPts1, kind='linear')
            return Tintfun(y)
    if (yintPts3[-1] < y < yintPts1[0]): # middle
        # if (len(yintPts2) == 1):
        # return TintPts2[0]
        # else:
        Tintfun = interpolate.interp1d(yy, Ty, kind='cubic', assume_sorted=
            False)
        return Tintfun(y)
    if (yintPts3[0] <= y <= yintPts3[-1]):
        if (len(yintPts3) == 1):
            return TintPts3[0]
        else:
            Tintfun = interpolate.interp1d(yintPts3, TintPts3, kind='linear')
            return Tintfun(y)


## @brief Finds temperature across different thermocouples based on the t.
# @details This method finds temperature across different thermocouples
    which can be used for interpolation. This makes sure that
# if the given time is less than the liq_time at that location, then only
    thermocouples which has temperature below liquidus at that time are
    considered.
# @param y - Location, t - time
# @exception none
# @return yintPts1,TintPts1(below liquidus point location and temp),
    yintPts2, TintPts2(2phase zone location and temp), yintPts3, TintPts3(
    after solidus point location and temp)
# @note Likely not needed
def _Tinterp(y, t):
    yy, Ty = slice(t)
    yyy, t1, t2 = breakPts()
    yintPts1 = []
    TintPts1 = []
    yintPts2 = []
    TintPts2 = []
    yintPts3 = []
    TintPts3 = []
    for i in range(len(yy)):
        if (t < t1[i]):
            yintPts1.append(yy[i])
            TintPts1.append(Ty[i])
        if (t1[i] <= t <= t2[i]):
            yintPts2.append(yy[i])
            TintPts2.append(Ty[i])
        if (t > t2[i]):
            yintPts3.append(yy[i])
            TintPts3.append(Ty[i])
    return yintPts1, TintPts1, yintPts2, TintPts2, yintPts3, TintPts3


## @brief Finding dT/dt
# @details This method finds dT/dt using forward difference formula. This
    method is used in Calculation module in various methods like
# calculate_h(), calculate_alpha() and calculate_FS.
# @param y - Location, t - time
# The details section needs modification once Calculation.py undergoes
    change.
# @exception none
# @see Configuration.py
# @return dT/dt
# @note This method is being used now.
def dTdt(y, t):
    # dt = Configuration.timestep
    dt = 0.0001 # Configuration.timestep #FIXME
    return (T(y, t + dt) - T(y, t)) / dt


## @brief Finding dT/dt
# @details This method finds dT/dt using fourth order centered difference
    formula. This method is used in Calculation module in various methods
    like
# calculate_h(), calculate_alpha() and calculate_FS.
# @param y - Location, t - time
# The details section needs modification once Calculation.py undergoes change.
# @exception none
# @see Configuration.py
# @return dT/dt
# @note This method is not used now.
def __dTdt(y, t): # higher order divided difference
    dt = 0.001 # Configuration.timestep #FIXME
    return (-T(y, t + 2 * dt) + 8 * T(y, t + dt) - 8 * T(y, t - dt) + T(y, t - 2 * dt)) / (12 * dt)


## @brief Compute curvature
# @details This method computes d2T/dy2 using central difference method at any given location and time. This uses configuration.py.
# @param y - Location, t - time
# @exception none
# @see Configuration.py
# @return d2Tdy2(number)
# @note Units have to checked (This was written below)
def d2Tdy2(y, t):
    # dy = Configuration.locstep
    dy = 0.01
    T1 = T(y + (3 * dy), t)
    T2 = T(y + (2 * dy), t)
    T3 = T(y + dy, t)
    T4 = T(y, t)

    return (-T1 + (4 * T2) - (5 * T3) + (2 * T4)) / (dy * dy)


## @brief Compute curvature
# @details This method computes d2T/dy2 using central difference method at any given location and time. This uses configuration.py.
# @param y - Location, t - time
# @exception none
# @see Configuration.py
# @return d2Tdy2(number)
# @note Units have to checked (This was written below)
def _d2Tdy2(y, t):
    # dy = Configuration.locstep
    dy = 0.01
    # print (T(y-dy, t))
    # print (T(y+dy, t))
    # print (T(y,t))
    return (T(y - dy, t) + T(y + dy, t) - 2 * T(y, t)) / (dy * dy)


## @brief Compute curvature
# @details This method computes d2T/dy2 using central difference method
   at any given location and time. This uses configuration.py.
# @param y - Location, t - time
# @exception none
# @see Configuration.py
# @return d2Tdy2(number)
# @note Units have to checked (This was written below)
def _d2Tdy2(y, t):
    dy = 0.01
    return (-T(y + (2 * dy), t) + 16 * T((y + dy), t) - 30 * T(t, y) + 6 * T
        ((y - dy), t) - T(y - (2 * dy), t) / 12 * (
                dy * dy))


## @brief Length of thermocouples which could be fit
# @details This method finds the no: of thermocouples which obtained a
   piecewise fit. This need not match the actual no: of thermocouples.
# @note Will be removed later. Not sure if it is used.
def size():
    return len(S)
```

D.2 Piecewise Module

```
## @file PiecewiseADT.py
# @author Vincent Lee, Spencer Smith and Malavika Srinivasan
# @brief Exports PiecewiseT - creates a Piecewise object which contains a
   piecewise polynomial fit for one thermocouple
```

```
# @date 17/04/2018

import numpy as np
from scipy import optimize
import math
## PiecewiseADT provides the curve fitting algorithm for
## thermocouples. The curve is made up of 3 segments (polynomials) with
    continuity
## of the value enforced, but no requirement for continuity of any
## derivatives. The algorithm may alter the order of the polynomials
## to achieve a better fit. Some details of the algorithm are
## available at @see{https://gitlab.cas.mcmaster.ca/smiths/sfs/blob/master/
    Doc/Design/MIS/MIS.pdf}



class PiecewiseADT:

    ## @brief PiecewiseT constructor
    # @details The constructor takes the values for the state
    # variables and sets them
    def __init__(self, ordP1, coeffP1, ordP2, coeffP2, ordP3, coeffP3,x1, x2,
        maxD, minD, foundFit):
      self._curFirstPolyDegree = ordP1
      self._p1 = coeffP1
      self._curSecondPolyDegree = ordP2
      self._p2 = coeffP2
      self._curThirdPolyDegree = ordP3
      self._p3 = coeffP3
      self._x1 = x1
      self._x2 = x2
      self._maxD = maxD
      self._minD = minD
      self._foundFit = foundFit

    ## @brief PiecewiseT fromPrevFit
    # @details Following the factory design pattern this class method
    # will return a PiecewiseADT object using previously determined
    # fit parameters (no need to refit)
    @classmethod
```

```
def fromPrevFit(cls, ordP1, coeffP1, ordP2, coeffP2, ordP3, coeffP3, x1,
    x2, maxD, minD, foundFit):
  # add [0] to match the raw data calculations - n degree poly
  # will have n+1 coefficients
  return cls(ordP1, coeffP1, ordP2, coeffP2+[0], ordP3, coeffP3+[0], x1,
      x2, maxD, minD, foundFit)



    ## @brief Piecewise Fitting
    # @details This method is used to obtain a piecewise fit for the
        thermocouple.
    # This follows the factory design pattern. This class method
# will return a PiecewiseADT object by using the raw temperature
# data and fitting the best polynomials.<br>
# This method is used by the Load.py to obtain a fit for the
    thermocouples.
    # @param xPoints - list of x values sorted in ascending order, yPoints
         - list of corresponding y values to the the x's in the x list,
    # x1_init - Initial guess for first breakpoint, x2_init - Initial guess
        for second breakpoint
    # @exception none
    # @return FirstPolyDegree, p1 (First polynomial degree, Polynomial
        coefficient)
    # curSecondPolyDegree, p2(Second polynomial degree, Polynomial
        coefficient),
    # curThirdPolyDegree, p3(Third polynomial degree, Polynomial
        coefficient),
    # x1, x2 (Optimized breakpoints),
    # maxD, minD - Minimum and maximum domain values (time values
         between which the fit is valid)
    # foundFit - True or False (This tells if a thermocouple has a fit or
        obtaining a fit was not feasible)
@classmethod
def fromRawData(cls, xPoints, yPoints, x1_init, x2_init):
    x1 = x1_init #First breakpoint
    x2 = x2_init #Second breakpoint
    maxD = 0 #max domain value
    minD = 0 #min domain value
    baseFirstPolyDegree = 5 #degree of first polynomial (min 1)
```

```
baseSecondPolyDegree = 4 #degree of second polynomial (min 1)
baseThirdPolyDegree = 6 #degree of third polynomial (min 2)
firstDegreeIncrease = 1 #how much to increase the degree of first
    poly todo: change this value from 0
secondDegreeIncrease = 1 #how much to increase the degree of
    second poly todo: change this value from 0
thirdDegreeIncrease = 1 #how much to increase the degree of third
    poly todo: change this value from 0
curFirstPolyDegree = baseFirstPolyDegree #current first poly degree
curSecondPolyDegree = baseSecondPolyDegree #current second poly
    degree
curThirdPolyDegree = baseThirdPolyDegree #current third poly
    degree
p1 = [0] * (baseFirstPolyDegree + 1) #Final Coefficents for the first
    polynomial (degree + constant, highest degree first)
p2 = [0] * (baseSecondPolyDegree + 1) #Final Coefficents for the
    second polynomial (degree + constant,highest degree first)
p3 = [0] * (baseThirdPolyDegree + 1) #Final Coefficents for the
    third polynomial (degree + constant,highest degree first)
initP1 = [0] * (baseFirstPolyDegree + 1) #Inital Coefficents for the
    first polynomial (degree + constant,highest degree first)
initP2 = [0] * (baseSecondPolyDegree + 1) #Inital Coefficents for the
     second polynomial (degree + constant,highest degree first)
initP3 = [0] * (baseThirdPolyDegree + 1) #Inital Coefficents for the
    third polynomial (degree + constant,highest degree first)
foundFit = False #flag to determine if a fit curve was found
print ('^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^')
print ('Printing␣x␣array␣from␣PiecewiseADT')
print (xPoints[0])
print ('Printing␣y␣array␣from␣PiecewiseADT')
print (yPoints[0])
print ('^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^')

#check if data matches in size
if len(xPoints) != len(yPoints):
    raise ValueError('Number␣of␣x-values␣do␣not␣match␣number␣of␣
        y-values')

#set min/max domain values
maxD = xPoints[-1]
```

```
minD = xPoints[0]

print('given␣x1␣=␣{0}␣and␣given␣x2=␣{1}\n'.format(x1, x2))

#get critical points of the 3rd polynomial curve
x4 = xPoints[-1]
x3 = round((x4 + x2)/2)
y4 = yPoints[indexOf(xPoints, x4)]
y3 = yPoints[indexOf(xPoints, x3)]

#first polynomial guess (assume linear, 1 degree)
y1 = yPoints[indexOf(xPoints, x1)]
initP1[baseFirstPolyDegree] = yPoints[0] #contsnt
initP1[baseFirstPolyDegree - 1] = (y1 - yPoints[0]) / x1 #first degree

#second polynomial guess (assume linear, 1 degree)
y2 = yPoints[indexOf(xPoints, x2)]
initP2[baseSecondPolyDegree] = y1 #constant
initP2[baseSecondPolyDegree - 1] = (y2 - y1) / (x2 - x1) #first
    degree

#third polynomial guess
initP3[baseThirdPolyDegree] = y2 #constant
initP3[baseThirdPolyDegree - 2] = ((y3 - y2) * (x4 - x2) - (y4 - y2)
    * (x3 - x2)) / (pow((x3 - x2), 2) * (x4 - x2) - pow((x4 - x2), 2) *
     (x3 - x2)) #second degree
initP3[baseThirdPolyDegree - 1] = (pow((y3 - y2), 2) * (y4 - y2) -
    pow((x4 - x2), 2) * (y3 - y2)) / (pow((x3 - x2), 2) * (x4 - x2) -
    pow((x4 - x2), 2) * (x3 - x2)) #first degree

#optimize the functions to create a final piecewise func
fitError = math.inf
fittedFirstDeg = 0
fittedSecondDeg = 0
fittedThirdDef = 0
for firstDegInc in range(0, firstDegreeIncrease + 1):
    for secondDegInc in range(0, secondDegreeIncrease + 1):
        for thirdDegInc in range(0, thirdDegreeIncrease + 1):
            testP1 = [0] * firstDegInc + initP1 # increase degrees if
                needed
```

```
testP2 = [0] * secondDegInc + initP2
testP3 = [0] * thirdDegInc + initP3

curFirstPolyDegree = baseFirstPolyDegree + firstDegInc
curSecondPolyDegree = baseSecondPolyDegree +
    secondDegInc
curThirdPolyDegree = baseThirdPolyDegree +
    thirdDegInc

pInit = [x1] + [x2] + testP1 + testP2[:-1] + testP3[:-1]
    #initial parameters

try:
    print('{0}␣and␣{1}␣and␣{2}\n'.format(
        curFirstPolyDegree, curSecondPolyDegree,
        curThirdPolyDegree))
    func = lambda x, x1, x2, *args: function(x, x1, x2,
        curFirstPolyDegree, curSecondPolyDegree,
        curThirdPolyDegree, *args)
    p, e = optimize.curve_fit(func, xPoints, yPoints,
        pInit, ftol=1e-15, maxfev = 5000)
    foundFit = True
except:
    continue

#check if error is less than best error
if math.fabs(sum(np.diag(e).tolist())) < fitError:

    fitError = math.fabs(sum(np.diag(e).tolist()))

    #adjust function coefficents based on curve fit results
    p = p.tolist()
    poly1EndIndex = 2 + (curFirstPolyDegree + 1) #2
        for first 2 params, +1 to include constant
    poly2EndIndex = poly1EndIndex +
        curSecondPolyDegree
    poly3EndIndex = poly2EndIndex +
        curThirdPolyDegree

    x1 = p[0]
```

```
                    x2 = p[1]
                    p1 = p[2:poly1EndIndex]
                    p2 = p[poly1EndIndex:poly2EndIndex] + [0]
                    p3 = p[poly2EndIndex:poly3EndIndex] + [0]

                    print('found␣fit␣and␣x1␣=␣{0}␣and␣x2=␣{1}\n'.
                        format(x1, x2))

                    fittedFirstDeg = curFirstPolyDegree
                    fittedSecondDeg = curSecondPolyDegree
                    fittedThirdDef = curThirdPolyDegree

    #at the end of fitting set the polynomial degrees
    curFirstPolyDegree = fittedFirstDeg
    curSecondPolyDegree = fittedSecondDeg
    curThirdPolyDegree = fittedThirdDef
    #print (foundFit)

    return cls(curFirstPolyDegree, p1, curSecondPolyDegree, p2,
        curThirdPolyDegree, p3, x1, x2, maxD, minD, foundFit)


## @brief minD returns the minimum value of the independent variable
# @return value representing the left limit of the indep var
def get_foundFit(self):
    return self._foundFit


    ## @brief feval evaluates function at a given x value.
    # @details This gives the information necessary to find the temp
        using the function().
    # This calls the function with the necessary parameters to find the
        temperature of the thermocouple at a particular time.
    # @param x - time value.
    # @exception Out of domain - Value error when x < minD and x >
        minD
    # @return Returns a function call to function() which in turn returns
         temperature at x.
def feval(self, x):
    #check if a fit was found
    if not self.__foundFit:
        raise ValueError('No␣fitting␣function␣found')

    #check if value is within domain
    if type(x) is np.ndarray:
        if np.any(x < self.__minD):
            raise ValueError('Out␣of␣domain')
        elif np.any( x > self.__maxD):
            raise ValueError('Out␣of␣domain')
    elif x < self.__minD or x > self.__maxD:
       raise ValueError('Out␣of␣domain')

    funcParams = self.__p1 + self.__p2[:-1] + self.__p3[:-1]
    return function(x, self.__x1, self.__x2, self.__curFirstPolyDegree, self.
        __curSecondPolyDegree, self.__curThirdPolyDegree,*funcParams)

## @brief minD returns the minimum value of the independent variable
# @return value representing the left limit of the indep var
def get_minD(self):
    return self.__minD

## @brief maxD returns the maximum value of the independent variable
# @return value representing the right limit of the indep var
def get_maxD(self):
    return self.__maxD

## @brief get__x1 returns the first breakpoint
# @return value representing the first breakpoint
def get__x1(self):
    return self.__x1

## @brief get__x2 returns the first breakpoint
# @return value representing the first breakpoint
def get__x2(self):
    return self.__x2

## @brief return coefficients
# @return values of coefficients for piecewise fit -
# @return returns each poly degree followed by the polynomial
    coefficients
    def get_coeff(self):
        ordP1 = self._curFirstPolyDegree
        coeffP1 = self._p1
        ordP2 = self._curSecondPolyDegree
        coeffP2 = self._p2[:-1]
        ordP3 = self._curThirdPolyDegree
        coeffP3 = self._p3[:-1]
        return ordP1, coeffP1, ordP2, coeffP2, ordP3, coeffP3


## @brief return coefficients
# @param list - array, value - element which has to be found.
# @return index of value closest to input value
def indexOf(list, value):
    return min(range(len(list)), key=lambda i: abs(list[i]-value))
```

Appendix E

Pre-development Interview

Interview Questions Before Software Development

Malavika Srinivasan and Spencer Smith

December 9, 2018

Contents

1 Introduction

This document contains a list of questions to assess the attitude of research scientists towards the development of a scientific software. Before beginning our document driven development, we want to assess the attitudes of the stakeholders. After the completion of the software development, we will interview them again to see if their attitudes have changed in any way as a result of our work.

2 Related Terminologies

3 Participants

P1: Dr. Sumanth Shankar - Professor, Dept of Mechanical Engineering, McMaster University.

P2: Dr. Mohammed Hamed - Professor, Dept of Mechanical Engineering, McMaster University.

P3: Dr. Kumar Sadayappan - Research Scientist, Natural Resources Canada - CanmetMATERIALS.

P4: Dr. Jeyakumar Manickaraj - Research Associate, Dept of Mechanical Engineering, McMaster University.

4 Questions

4.1 General Questions

Q1: Please let us know your scientific background. This information will be helpful in evaluating our results when we are writing up the thesis.

A1: Phd in Material science and Engg from Worcester Polytechnic Institute,Massachusetts. UnderGrad in Metallurgy from- IIT.

Q2: What is your level of programming experience? What programming languages have you used?

A2: Have some programming experience in Fortran, Pascal, Visual basic, Matlab, Machine level logic, definition of algorithms e.t.c. No programming courses were taken. Syntax of any programming language needs little more experience. Can trouble shoot and understand code in C, C++, Fortran.

Q3: What software tools do you use in the course of doing your scientific work?

A3: Tools used include - Microsoft suite. Latex was used and discontinued. Graphical softwares like easy plot(extremely user friendly and condensed), Familiar with

Sigma plot, Pandat(Thermodynamic software), Factsage , ThermoCalc e.t.c. Have some trainings in CAD and ProE. Have good analysis knowledge in Photoshop. Version control software or issue tracker was not used. GIT will be used.

Q4: Do you have an initial thoughts on how one should go about developing high quality scientific software?

A4: Based on personal experience a good software should be able to have a help system to answer questions if anything needs to be done. It should have a help tutorial to learn a software. Dynamic tutorials like user documentation would be even better.

Q5: In the SFS project what are some ways that the software could fail? What is the worst that could happen if the software fails?

A5: There are no safety concerns if the project fails. There is no wastage of money compared to the size and total budget of the project. If this project fails, the calculations done by this software will be done manually but that would be tedious.

Q6: What if the software works well but doesn?t give correct results?

A6: Then it still would be good data because it is useful in finding whether the understanding about the science behind this project is wrong or implementation of the science in the software is wrong.

Q7: What are the most important software qualities for your work?

A7: Correctness is essential because significantly important industries will be the users of this software and lots of money will be involved to take it to industrial scale and these industries act as the backbone of economy in many nations. Verifiability will be important as well . Speed is important, the maximum timeline is 15 min to max 30 min altogether for the casting experiment, data acquisition by a DAQ, the software to work and then give a GO or no GO. Portability is not very important. Maintainability is required as it is continuous evolution.

Q8: What are your attitudes toward documentation of a software product? What kind of information do you think should be documented?

A8: Documentation should be a combination of both software and scientific aspects. It shouldn't be long. It could be a substitute for textbook references. Can contribute some document content like theory and other scientific aspects.

Q9: Do you use other people's software? How do you ensure that it is trustworthy? What do you find is the greatest difficulty in using other people's software? Ideally, what would you like to see that would help?

A9: Just trust the software

Q10: What could have other softwares given you that could have helped?

A10: Test cases can be generated and added as appendix to documentation.. Test cases are important in aerospace and automobile industries. When castings are modelled, the software can be used to predict fraction solid and that could be a verification system for the SW. (Can be included in VV plan)

Appendix F

Guide for developing SC Software

In this document we provide a set of guidelines based on our experience in developing a scientific software.

1. At first, discuss with the scientists and identify the key requirements and likely changes.
2. Identify the properties of correct input and output data. This information can be used to automate the software to test for metamorphic trends in the input and output data which can be used as a preliminary test before processing the data.
3. Adopt a 'Document driven design' in combination with 'Faked rational process design' to present the documents.
4. Involve scientists to a maximum extent from the beginning of the development process. If possible, have a partner in coding who is also a domain scientist. It is better to avoid SE jargons when communicating with the scientists.
5. "Design for change" must be adopted for SC.
6. When finalizing the mathematical equations used in the software, convert them to canonical forms so that any generic solvers can be used.
7. When implementing the software, choose a programming language based on the requirements of the project and not based on scientists' familiarity of programming languages.
8. Identify the test cases for each module and start testing the modules as they are implemented. However, it is not easy to identify test cases for a SC application due to the oracle problem. Hence, the developer needs to be adept and insightful in identifying the test cases.
9. If the software does not produce desirable results, it is important to concentrate on testing the modules to improve confidence in coding.
10. If trial and error experiments are necessary, adopt a SE approach such as regression testing, which provides a way to measure each approach. It is also advisable to keep track of a log in issue tracker where the list of all approaches along with their results can be maintained. A version control system such as Git may be used in which trials can be conducted on a separate branches and the best approach can be used by merging the branch.
11. If document reviews are necessary, adopt 'Task based inspection' which requires focused effort, meaning lesser time commitment from the scientists.

12. In general, easy to use, lazy proof tools are readily accepted by the scientists. Making proper use of such tools will be helpful during the development process.