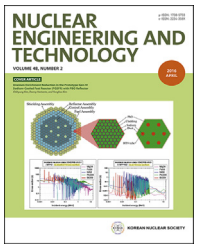
Nuclear Engineering and Technology 48 (2016) 404–418

Available online at ScienceDirect

Nuclear Engineering and Technology

journal homepage: www.elsevier.com/locate/net



Original Article

A Document-Driven Method for Certifying Scientific Computing Software for Use in Nuclear Safety Analysis

W. Spencer Smith[*,1] *and Nirmitha Koothoor*

Computing and Software Department, McMaster University, Hamilton, Ontario L8S 4L7, Canada

ARTICLE INFO

Article history:
Received 12 June 2015
Received in revised form
16 November 2015
Accepted 29 November 2015
Available online 18 December 2015

Keywords:
Literate Programming
Nuclear Safety Analysis
Numerical Simulation
Requirements Specification
Software Engineering
Software Quality

ABSTRACT

This paper presents a documentation and development method to facilitate the certification of scientific computing software used in the safety analysis of nuclear facilities. To study the problems faced during quality assurance and certification activities, a case study was performed on legacy software used for thermal analysis of a fuelpin in a nuclear reactor. Although no errors were uncovered in the code, 27 issues of incompleteness and inconsistency were found with the documentation. This work proposes that software documentation follow a rational process, which includes a software requirements specification following a template that is reusable, maintainable, and understandable. To develop the design and implementation, this paper suggests literate programming as an alternative to traditional structured programming. Literate programming allows for documenting of numerical algorithms and code together in what is termed the literate programmer's manual. This manual is developed with explicit traceability to the software requirements specification. The traceability between the theory, numerical algorithms, and implementation facilitates achieving completeness and consistency, as well as simplifies the process of verification and the associated certification.

1. Introduction

This paper focuses on the certification of scientific computing (SC) software used for safety analysis in the design of nuclear facilities. Although this class of software is not considered as safety-critical, since it does not control the operation of a nuclear reactor or the associated safety systems, high-quality SC software is necessary for designing efficient and safe power plants. Standards and guidelines exist for producing SC software in a nuclear context, such as the Canadian requirements for quality assurance (QA) of scientific computer programs [1–3] and the US Department of Energy (DOE) guidelines for determining the adequacy of software used in safety analysis and design [4]. These publications list documentation that is expected for QA activities, including software requirements, design specification and verification, and

* *Corresponding author.*
E-mail address: smiths@mcmaster.ca (W.S. Smith).
[1] URL: http://www.cas.mcmaster.ca/%7esmiths/
http://dx.doi.org/10.1016/j.net.2015.11.008

validation reports. The standards and guidelines lay out at a high (abstract) level of what needs to be achieved by documentation, but at times they give limited concrete information on how to achieve these requirements. This paper fills in the missing details by proposing a systematic method for writing complete, consistent, and verifiable documentation for SC software used in nuclear safety analysis.

For certification to be successful, the documentation and code should have the qualities of verifiability, validatability, reliability, usability, maintainability, reusability, and reproducibility. With the exception of reproducibility and validatability, these qualities for general software are defined in a report by Ghezzi et al. [5, pp. 18–28]. In a SC context, verification means "solving the equations right" and validatability is "solving the right equations" [6, p. 23]. Reproducibility means being able to rerun the code in the future, possibly through an independent third party, and obtaining identical results [7].

Maintainability is necessary in SC, because change through iteration, experimentation and exploration is inevitable. Models of physical phenomena necessarily evolve over time [8,9], as do the numerical techniques used to simulate the models. QA activities need to take this need for creativity into account without smothering it [6, p. 352]. Maintainability is of practical importance because when changes occur after the initial certification, the recertification process must be significantly easier and cheaper than the first certification exercise, or recertification is unlikely to happen. Similarly, reusability is important for certification, because reuse can save time and money spent on the certification of similar products by reusing trusted components [10]. Fortunately, SC software is well suited for reuse, as program families (sets of programs where there are nontrivial commonalities and predictable variabilities) are frequently encountered in SC [11].

Documentation for nuclear safety software [2,4] follows the typical stages of the waterfall model of software development, as shown in Fig. 1. Given the exploratory nature of SC, developers do not follow this waterfall model [12,13], but this is not a problem for the documentation. As Parnas and Clements [14] point out, the most logical way to present the documentation is to "fake" a rational design process. "Software manufacturers can define their own internal process as long as they can effectively map their products onto the ones that the much simpler, faked process requires" [15]. To keep the scope of the current work manageable, we focus on the three middle stages from Fig. 1: requirements, design, and code implementation.

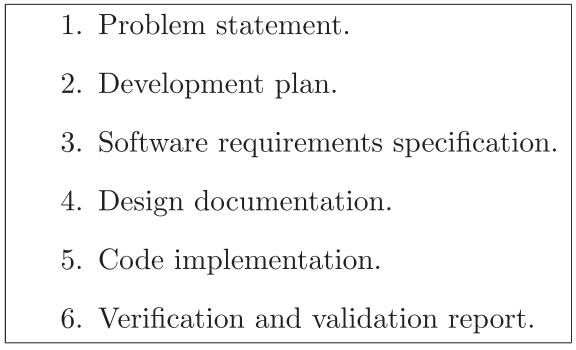


Fig. 1 – Software documentation following a rational process.

To develop, test, and justify the documentation and development method proposed, we conducted a case study with existing SC software, for which QA and the associated documentation are important considerations. The case study uses legacy nuclear safety analysis software provided by a power generation company. The software under study performs thermal analysis of a single fuelpin in a nuclear reactor by simulating simplified reactor physics and fuel management calculations. In the discussion that follows, the software will be referred to as FP. Along with the source code for FP we also received a theory manual, which includes the requirements, numerical algorithms, assumptions, constraints, and the mathematical model.

Our approach, which was described by Koothoor [16] and Smith et al. [17], was to redo the thermal analysis portion of the original FP code using modern software engineering techniques. By redoing the previous work, we were able to judge whether there is room for improvement and then propose a new and improved process. The design and development of the new documentation was done to be consistent with the Canadian standard for quality assurance of analytical, scientific, and design computer programs for nuclear power plants, N286.7, clause 11.2 [2]. Although the conclusions from this paper are based on the case study, the case study is considered representative of many other SC programs.

Section 2 provides background on the software engineering methods that are employed in the documentation, namely, software requirements specification (SRS) and literate programming (LP). This background section also gives an overview of the FP case study. Section 3 provides examples for the SRS for FP, along with an evaluation of the improvements of the new documentation compared with the old. Section 4 presents the LP excerpts from FP and explains how the literate programmer's manual (LPM) contributes to the goal of producing certifiable documentation. The final section, Section 5, consists of concluding remarks.

2. Background

How do we create documentation that facilitates achieving qualities such as verifiability and maintainability? Unfortunately, these qualities are examples of ones that cannot be measured directly. They must be measured indirectly, since their measurement depends on interactions with the environment [18, p. 109]. Moreover, many of the qualities being considered, such as reliability and usability, are external qualities, which means that they are measured by their impact on the user, as opposed to the software developer [5, p. 16]. Although the user and the developer in SC are often the same person, here we are making the distinction based on the role of the individual. With their connection to the end user, external qualities can only be measured when the software is complete. We need internal measures that can be assessed as the software is being built, so that we have confidence that we are on the right track for success. We also need measures that have a smaller scope, so that their measurement is not so