Information and Software Technology 56 (2014) 1219–1232

Contents lists available at ScienceDirect

Information and Software Technology

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

Testing scientific software: A systematic literature review

Upulee Kanewala *, James M. Bieman

Computer Science Department, Colorado State University, USA

ARTICLE INFO

Article history:
Received 14 October 2013
Received in revised form 12 May 2014
Accepted 14 May 2014
Available online 23 May 2014

Keywords:
Scientific software
Software testing
Systematic literature review
Software quality

ABSTRACT

Context: Scientific software plays an important role in critical decision making, for example making weather predictions based on climate models, and computation of evidence for research publications. Recently, scientists have had to retract publications due to errors caused by software faults. Systematic testing can identify such faults in code.
Objective: This study aims to identify specific challenges, proposed solutions, and unsolved problems faced when testing scientific software.
Method: We conducted a systematic literature survey to identify and analyze relevant literature. We identified 62 studies that provided relevant information about testing scientific software.
Results: We found that challenges faced when testing scientific software fall into two main categories: (1) testing challenges that occur due to characteristics of scientific software such as oracle problems and (2) testing challenges that occur due to cultural differences between scientists and the software engineering community such as viewing the code and the model that it implements as inseparable entities. In addition, we identified methods to potentially overcome these challenges and their limitations. Finally we describe unsolved challenges and how software engineering researchers and practitioners can help to overcome them.
Conclusions: Scientific software presents special challenges for testing. Specifically, cultural differences between scientist developers and software engineers, along with the characteristics of the scientific software make testing more difficult. Existing techniques such as code clone detection can help to improve the testing process. Software engineers should consider special challenges posed by scientific software such as oracle problems when developing testing techniques.

Contents

* Corresponding author. Tel.: +1 9704917096.
E-mail addresses: upuleegk@cs.colostate.edu (U. Kanewala), bieman@cs.colostate.edu (J.M. Bieman).

http://dx.doi.org/10.1016/j.infsof.2014.05.006
0950-5849/

1. Introduction

Scientific software is widely used in science and engineering fields. Such software plays an important role in critical decision making in fields such as the nuclear industry, medicine and the military [65,66]. For example, in nuclear weapons simulations, code is used to determine the impact of modifications, since these weapons cannot be field tested [62]. Climate models make climate predictions and assess climate change [17]. In addition, results from scientific software are used as evidence in research publications [66]. Due to the complexity of scientific software and the required specialized domain knowledge, scientists often develop these programs themselves or are closely involved with the development [60,47,69,7]. But scientist developers may not be familiar with accepted software engineering practices [69,65]. This lack of familiarity can impact the quality of scientific software [20].

Software testing is one activity that is impacted. Due to the lack of systematic testing of scientific software, subtle faults can remain undetected. These subtle faults can cause program output to change without causing the program to crash. Software faults such as one-off errors have caused the loss of precision in seismic data processing programs [27]. Software faults have compromised coordinate measuring machine (CMM) performance [1]. In addition, scientists have been forced to retract published work due to software faults [51]. Hatton et al. found that several software systems written for geoscientists produced reasonable yet essentially different results [28]. There are reports of scientists who believed that they needed to modify the physics model or develop new algorithms, but later discovered that the real problems were small faults in the code [18].

We define scientific software broadly as *software used for scientific purposes*. Scientific software is mainly developed to better understand or make predictions about real world processes. The size of this software ranges from 1,000 to 100,000 lines of code [66]. Developers of scientific software range from scientists who do not possess any software engineering knowledge to experienced professional software developers with considerable software engineering knowledge.

To develop scientific software, scientists first develop discretized models. These discretized models are then translated into algorithms that are then coded using a programming language. Faults can be introduced during all of these phases [15]. Developers of scientific software usually perform validation to ensure that the scientific model is correctly modeling the physical phenomena of interest [37,57]. They perform verification to ensure that the computational model is working correctly [37], using primarily mathematical analyses [62]. But scientific software developers rarely perform systematic testing to identify faults in the code [38,57,32,65]. Farrell et al. show the importance of performing code verification to identify differences between the code and the discretized model [24]. Kane et al. found that automated testing is fairly uncommon in biomedical software development [33]. In addition, Reupke et al. discovered that many of the problems found in operational medical systems are due to inadequate testing [64]. Sometimes this lack of systematic testing is caused by special testing challenges posed by this software [20].

This work reports on a Systematic Literature Review (SLR) that identifies the special challenges posed by scientific software and proposes solutions to overcome these challenges. In addition, we identify unsolved problems related to testing scientific software.

An SLR is a "means of evaluating and interpreting all available research relevant to a particular research question or topic area or phenomenon of interest" [41]. The goal of performing an SLR is to methodically review and gather research results for a specific research question and aid in developing evidence-based guidelines for the practitioners [42]. Due to the systematic approach followed when performing an SLR, the researcher can be confident that she has located the required information as much as possible.

Software engineering researchers have conducted SLRs in a variety of software engineering areas. Walia et al. [77] conducted an SLR to identify and classify software requirement errors. Engström et al. [23] conducted an SLR on empirical evaluations of regression test selection techniques with the goal of "finding a basis for further research in a joint industry-academia research project". Afzal et al. [3] carried out an SLR on applying search-based testing for performing non-functional testing. Their goal is to "examine existing work into non-functional search-based software testing". While these SLRs are not restricted to software in a specific domain, we focus on scientific software, an area that has received less attention than application software. Further when compared to Engström et al. or Afzal et al., we do not restrict our SLR to a specific testing technique.

The overall goal [42] of our SLR is to *identify specific challenges faced when testing scientific software, how the challenges have been met, and any unsolved challenges.* We developed a set of research questions based on this overall goal to guide the SLR process. Then we performed an extensive search to identify publications that can help to answer these research questions. Finally, we synthesized the gathered information from the selected studies to provide answers to our research questions.

This SLR identifies two categories of challenges in scientific software testing. The first category are challenges that are due to the characteristics of the software itself such as the lack of an oracle. The second category are challenges that occur because scientific software is developed by scientists and/or scientists play leading roles in scientific software development projects, unlike application software development where software engineers play leading roles. We identify techniques used to test scientific software including techniques that can help to overcome oracle problems and test case creation/selection challenges. In addition, we describe the limitations of these techniques and open problems.

This paper is organized as follows: Section 2 describes the SLR process and how we apply it to find answer to our research questions. We report the findings of the SLR in Section 3. Section 4 contains the discussion on the findings. Finally we provide conclusions and describe future work in Section 5.

2. Research method

We conducted our SLR following the published guidelines by Kitchenham [41]. The activities performed during an SLR can be