Published in final edited form as:
Inf Softw Technol. 2014 October 1; 56(10): 1219–1232. doi:10.1016/j.infsof.2014.05.006.

Testing Scientific Software: A Systematic Literature Review

Upulee Kanewala[*] and **James M. Bieman**
Computer Science Department, Colorado State University, USA

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.

Keywords

Scientific software; Software testing; Systematic literature review; Software quality

[*]Corresponding author upuleegk@cs.colostate.edu (Upulee Kanewala), bieman@cs.colostate.edu (James M. Bieman).

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. Engstrom *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 Engstrom *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 divided into three main phases: (1) planning the

SLR, (2) conducting the review and (3) reporting the review. We describe the tasks performed in each phase below.

2.1. Planning the SLR

2.1.1. Research Questions—The main goal of this SLR is to identify specific challenges faced when testing scientific software, how the challenges have been met, and any unsolved challenges. We developed the following research questions to achieve our high level goal:

RQ1: How is scientific software defined in the literature?

RQ2: Are there special characteristics or faults in scientific software or its development that make testing difficult?

RQ3: Can we use existing testing methods (or adapt them) to test scientific software effectively?

RQ4: Are there challenges that could not be met by existing techniques?

2.1.2. Formulation and validation of the review protocol—The review protocol specifies the methods used to carry out the SLR. Defining the review protocol prior to conducting the SLR can reduce researcher bias [43]. In addition, our review protocol specifies source selection procedures, search process, quality assessment criteria and data extraction strategies.

Source selection and search process: We used the Google Scholar, IEEE Xplore, and ACM Digital Library databases since they include journals and conferences focusing on software testing as well as computational science and engineering. Further, these databases provide mechanisms to perform key word searches. We did not specify a fixed time frame when conducting the search. We conducted the search in January 2013. Therefore this SLR includes studies that were published before January 2013. We did not search for specific journals/conferences since an initial search found relevant studies published in journals such as Geoscientific Model Development[1] that we were not previously familiar with. In addition, we examined relevant studies that were referenced by the selected primary studies.

We searched the three databases identified above using a search string that included the important key words in our four research questions. Further, we augmented the key words with their synonyms, producing the following search string:

(((challenges **OR** problems **OR** issues **OR** characteristics) **OR** (technique **OR** methods **OR** approaches)) **AND** (test **OR** examine)) **OR** (error **OR** fault **OR** defect **OR** mistake **OR** problem **OR** imperfection **OR** flaw **OR** failure) **AND** ("(scientific **OR** numeric **OR** mathematical **OR** floating point) **AND** (Software **OR** application **OR** program **OR** project **OR** product)")

[1]http://www.geoscientic-model-development.net/

Study selection procedure: We systematically selected the primary studies by applying the following three steps.

1. We examined the paper titles to remove studies that were clearly unrelated to our search focus.
2. We reviewed the abstracts and key words in the remaining studies to select relevant studies. In some situations an abstract and keywords did not provide enough information to determine whether a study is relevant. In such situations, we reviewed the conclusions.
3. We filtered the remaining studies by applying the inclusion/exclusion criteria given in Table 1. Studies selected from this final step are the initial primary studies for the SLR.

We examined the reference lists of the initial primary studies to identify additional studies that are relevant to our search focus.

Quality assessment checklist: We evaluated the quality of the selected primary studies using selected items from the quality checklists provided by Kitchenham and Charters [43]. Table 2 and Table 3 show the quality checklists that we used for quantitative and qualitative studies respectively. When creating the quality checklist for quantitative studies, we selected quality questions that would evaluate the four main stages of a quantitative study: design, conduct, analysis and conclusions [43].

Data extraction strategy: Relevant information for answering the research questions needed to be extracted from the selected primary studies. We used data extraction forms to make sure that this task was carried out in a accurate and consistent manner. Table 4 shows the data extraction from that we used.

2.2. Conducting the review

2.2.1. Identifying relevant studies and primary studies—The key word based search produced more than 6000 hits. We first examined paper titles to remove any studies that are not clearly related to the research focus. Then we used the abstract, key words and the conclusion to eliminate additional unrelated studies. After applying these two steps, 94 studies remained. We examined these 94 studies and applied the inclusion/exclusion criteria in Table 1 to select 49 papers as primary studies for this SLR.

Further, we applied the same selection steps to the reference lists of the selected 49 primary studies to find additional primary studies that are related to the research focus. We found 13 studies that are related to our research focus that were not already included in the initial set of primary studies. Thus, we used a total of 62 papers as primary studies for the SLR. The selected primary studies are listed in Tables 5 and 6. Table 7 lists the publication venues of the selected primary papers. The International Workshop on Software Engineering for Computational Science and Engineering and the Journal of Computing in Science & Engineering published the greatest number of primary studies.

2.2.2. Data extraction and quality assessment

We used the data extraction form in Table 4 to extract data from the primary studies. Many primary studies did not answer all of the questions in the data extraction form. We extracted the important information provided by the primary studies using the data extraction form. Then, depending on the type of the study, we applied the quality assessment questions in Table 2 or Table 3 to each primary study.

We provided ‘yes’ and ‘no’ answers to our quality assessment questions. We used a binary scale since we were not interested in providing a quality score for the studies [19]. Table 8 shows the results of the quality assessment for quantitative primary studies. All the quantitative primary studies answered ‘yes’ to the quality assessment question G1 (Are the study aims clearly stated?). Most of the quantitative primary studies answered ‘yes’ to the quality assessment questions G2 (Are the data collection methods adequately described) and G5 (Can the study be replicated?). Table 9 shows the results of the quality assessment for qualitative primary studies. All of the qualitative primary studies answered ‘yes’ to the quality assessment question A (Are the study aims clearly stated?) and B (Does the evaluation address its stated aims and purpose?). Most of the qualitative primary studies answered ‘yes’ to the quality assessment question D (Is enough evidence provided to support the claims?).

2.3. Reporting the review

Data extracted from the 62 primary papers were used to formulate answers to the four research questions given in Section 2.1.1. We closely followed guidelines provided by Kitchenham [41] when preparing the SLR report.

3. Results

We use the selected primary papers to provide answers to the research questions.

3.1. RQ1: How is scientific software defined in the literature?

Scientific software is defined in various ways. Sanders *et al*. [65] use the definition provided by Kreyman *et al*. [44]: “Scientific software is software with a large computational component and provides data for decision support.” Kelly *et al*. identified two types of scientific software [39]:

1. End user application software that is written to achieve scientific objectives (e.g., Climate models).
2. Tools that support writing code that express a scientific model and the execution of scientific code (e.g., Automated software testing tool for MATLAB [22]).

An orthogonal classification is given by Carver *et al*. [8]:

1. Research software written with the goal of publishing papers.
2. Production software written for real users (e.g. Climate models).

Scientific software is developed by scientists themselves or by multi-disciplinary teams, where a team consists of scientists and professional software developers. A scientist will generally be the person in charge of a scientific software development project [53].

We encountered software that helps to solve a variety of scientific problems. We present the details of software functionality, size and the programing languages in Table 10. None of the primary studies reported the complexity of the software in terms of measurable unit such as coupling, cohesion, or cyclomatic complexity.

3.2. RQ2: Are there special characteristics or faults in scientific software or its development that make testing difficult?

We found characteristics that fall into two main categories 1) Testing challenges that occur due to characteristics of scientific software, and 2) Testing challenges that occur due to cultural differences between scientists and the software engineering community. Below we describe these challenges:

1. Testing challenges that occur due to characteristics of scientific software: These challenges can be further categorized according to the specific testing activities where they pose problems.

    a. Challenges concerning test case development:

        i. Identifying critical input domain boundaries *a priori* is difficult due to the complexity of the software, round-off error effects, and complex computational behavior. This makes it difficult to apply techniques such as equivalence partitioning to reduce the number of test cases [66, 36, 7].

        ii. Manually selecting a sufficient set of test cases is challenging due to the large number of input parameters and values accepted by some scientific software [76].

        iii. When testing scientific frameworks at the system level, it is difficult to choose a suitable set of test cases from the large number of available possibilities [63].

        iv. Some scientific software lacks real world data that can be used for testing [55].

        v. Execution of some paths in scientific software are dependent on results of floating point calculations. Finding test data to execute such program paths is challenging [5].

        vi. Some program units (functions, subroutines, methods) in scientific software contain so many decisions that testing is impractical [52].

        vii. Difficulties in replicating the physical context where the scientific code is suppose to work can make comprehensive testing impossible [67].

b. Challenges towards producing expected test case output values (Oracle problems): Software testing requires an oracle, a mechanism for checking whether the program under test produces the expected output when executed using a set of test cases. Obtaining reliable oracles for scientific programs is challenging [65]. Due to the lack of suitable oracles it is difficult to detect subtle faults in scientific code [37]. The following characteristics of scientific software make it challenging to create a test oracle:

  i. Some scientific software is written to find answers that are previously unknown. Therefore only approximate solutions might be available [20, 57, 78, 7, 39].

  ii. It is difficult to determine the correct output for software written to test scientific theory that involves complex calculations or simulations. Further, some programs produce complex outputs making it difficult to determine the expected output [73, 65, 54, 11, 38, 61, 26, 78, 70].

  iii. Due to the inherent uncertainties in models, some scientific programs do not give a single correct answer for a given set of inputs. This makes determining the expected behavior of the software a difficult task, which may depend on a domain expert's opinion [1].

  iv. Requirements are unclear or uncertain up-front due to the exploratory nature of the software. Therefore developing oracles based on requirements is not commonly done [73, 59, 26, 30].

  v. Choosing suitable tolerances for an oracle when testing numerical programs is difficult due to the involvement of complex floating point computations [61, 36, 40, 12].

c. Challenges towards test execution:

  i. Due to long execution times of some scientific software, running a large number of test cases to satisfy specific coverage criteria is not feasible [36].

d. Challenges towards test result interpretation:

  i. Faults can be masked by round-off errors, truncation errors and model simplifications [36, 28, 26, 9, 12].

  ii. A limited portion of the software is regularly used. Therefore, less frequently used portions of the code may contain unacknowledged errors [60, 47].

  iii. Scientific programs contain a high percentage of duplicated code [52].

2. Testing challenges that occur due to cultural differences between scientists and the software engineering community: Scientists generally play leading roles in developing scientific software.

    a. Challenges due to limited understanding of testing concepts:

        i. Scientists view the code and the model that it implements as inseparable entities. Therefore they test the code to assess the model and not necessarily to check for faults in the code [38, 47, 66, 65].

        ii. Scientist developers focus on the scientific results rather than the quality of the software [21, 7].

        iii. The value of the software is underestimated [70].

        iv. Definitions of verification and validation are not consistent across the computational science and engineering communities [32].

        v. Developers (scientists) have little or no training in software engineering [20, 21, 26, 7, 8].

        vi. Requirements and software evaluation activities are not clearly defined for scientific software [69, 71].

        vii. Testing is done only with respect to the initial specific scientific problem addressed by the code. Therefore the reliability of results when applied to a different problem cannot be guaranteed [53].

        viii. Developers are unfamiliar with testing methods [22, 26].

    b. Challenges due to limited understanding of testing process

        i. Management and budgetary support for testing may not be provided [59, 30, 71].

        ii. Since the requirements are not known up front, scientists may adopt an agile philosophy for development. However, they do not use standard agile process models [21]. As a result, unit testing and acceptance testing are not carried out properly.

        iii. Software development is treated as a secondary activity resulting in a lack of recognition for the skills and knowledge required for software development [68].

        iv. Scientific software does not usually have a set of written or agreed set of quality goals [52].

        v. Often only ad-hoc or unsystematic testing methods are used [65, 68].

   - **vi.** Developers view testing as a task that should be done late during software development [29].

- **c.** Challenges due to not applying known testing methods

   - **i.** The wide use of FORTRAN in the scientific community makes it difficult to utilize many testing tools from the software engineering community [47, 66, 21].

   - **ii.** Unit testing is not commonly conducted when developing scientific software [79, 18]. For example, Clune *et al*. find that unit testing is almost non-existent in the climate modeling community [12]. Reasons for the lack of unit testing include the following:

      - There are misconceptions about the difficulty and benefits of implementing unit tests among scientific software developers [12].
      - The legacy nature of scientific code makes implementing unit tests challenging [12].
      - The internal code structure is hidden [75].
      - The importance of unit testing is not appreciated by scientist developers [72].

   - **iii.** Scientific software developers are unaware of the need for and the method of applying verification testing [65].

   - **iv.** There is a lack of automated regression and acceptance testing in some scientific programs [8].

The following specific faults are reported in the selected primary studies:

- Incorrect use of a variable name [9].
- Incorrectly reporting hardware failures as faults due to ignored exceptions [52].
- One-off errors [27].

3.3. RQ3: Can we use existing testing methods (or adapt them) to test scientific software effectively?

Use of testing at different abstraction levels and for different testing purposes—Several primary studies reported conducting testing at different abstraction levels: unit testing, integration testing and system testing. In addition some studies reported the use of acceptance testing and regression testing. Out of the 62 primary studies, 12 studies applied at least one of these testing methods. Figure 1 shows the percentage of studies that applied each testing method out of the 12 studies. Unit testing was the most common testing method reported among the 12 studies.

Figure 2 displays the percentage of the number of testing methods applied by the 12 studies. None of the studies applied four or more testing methods. Out of the 12 studies, 8 (67%) mention applying only one testing method. Below we describe how these testing methods were applied when testing scientific software:

1. Unit testing: Several studies report that unit testing was used to test scientific programs [33, , 17, 2, 40, 45]. Clune *et al*. describe the use of refactoring to extract testable units when conducting unit testing on legacy code [12]. They identified two faults using unit testing that could not be discovered by system testing. Only two studies used a unit testing framework to apply automated unit testing [33, 2] and both of these studies used JUnit[2]. In addition, Eddins [22] developed a unit testing framework for MATLAB. We did not find evidence of the use of any other unit testing frameworks.
2. Integration testing: We found only one study that applied integration testing to ensure that all components work together as expected [17].
3. System testing: Several studies report the use of system testing [33, 24, 64]. In particular, the climate modeling community makes heavy use of system testing [12].
4. Acceptance testing: We found only one study that reports on acceptance testing conducted by the users to ensure that programmers have correctly implemented the required functionality [33]. One reason acceptance testing is rarely used is that the scientists who are developing the software are often also the users.
5. Regression testing: Several studies describe the use of regression testing to compare the current output to previous outputs to identify faults introduced when the code is modified [24, 17, 31, 75]. Further, Smith developed a tool for assisting regression testing [74]. This tool allows testers to specify the variable values to be compared and tolerances for comparisons.

Techniques used to overcome oracle problems—Previously we described several techniques used to test programs that do not have oracles [35]. In addition, several studies propose techniques to alleviate the oracle problem:

1. A *pseudo oracle* is an independently developed program that fulfills the same specification as the program under test [1, 59, 24, 21, 62, 65, 78, 16, 27]. For example, Murphy *et al*. used pseudo oracles for testing a machine learning algorithm [54].

   Limitations: A pseudo oracle may not include some special features/treatments available in the program under test and it is difficult to decide whether the oracle or the program is faulty when the answers do not agree [9]. Pseudo oracles make the assumption that independently developed reference models will not result in the same failures. But Brilliant *et al*. found that even independently developed programs might produce the same failures [6].

[2]http://junit.org/

2. Solutions obtained analytically can serve as oracles. Using analytical solutions is sometimes preferred over pseudo oracles since they can identify common algorithmic errors among the implementations. For example, a theoretically calculated rate of convergence can be compared with the rate produced by the code to check for faults in the program [1, 38, 24].

   Limitations: Analytical solutions may not be available for every application [9] and may not be accurate due to human errors [65].

3. Experimentally obtained results can be used as oracles [1, 38, 59, 62, 65, 45].

   Limitations: It is difficult to determine whether an error is due to a fault in the code or due to an error made during the model creation [9]. In some situations experiments cannot be conducted due to high cost, legal or safety issues [7].

4. Measurements values obtained from natural events can be used as oracles.

   Limitations: Measurements may not be accurate and are usually limited due to the high cost or danger involved in obtaining them [38, 66].

5. Using the professional judgment of scientists [66, 40, 32, 65]

   Limitations: Scientists can miss faults due to misinterpretations and lack of data. In addition, some faults can produce small changes in the output that might be difficult to identify [32]. Further, the scientist may not provide objective judgments [65].

6. Using simplified data that so the correctness can be determined easily [78].

   Limitations: It is not sufficient to test using only simple data; simple test cases may not uncover faults such as round-off problems, truncation errors, overflow conditions, etc [31]. Further such tests do not represent how the code is actually used [65].

7. Statistical oracle: verifies statistical characteristics of test results [49].

   Limitations: Decisions by a statistical oracle may not always be correct. Further a statistical oracle cannot decide whether a single test case has passed or failed [49].

8. Reference data sets: Cox *et al*. created reference data sets based on the functional specification of the program that can be used for black-box testing of scientific programs [13].

   Limitations: When using reference data sets, it is difficult to determine whether the error is due to using unsuitable equations or due to a fault in the code.

9. *Metamorphic testing (MT)* was introduced by Chen *et al*. [10] as a way to test programs that do not have oracles. MT operates by checking whether a program under test behaves according to an expected set of properties known as *metamorphic relations*. A metamorphic relation specifies how a particular change to the input of the program should change the output. MT was used for testing scientific applications in different areas such as machine learning applications [80, 56], bioinformatics programs [11], programs solving partial differential equations

[9] and image processing applications [48]. When testing programs solving partial differential equations, MT uncovered faults that cannot be uncovered by special value testing [9]. MT can be applied to perform both unit testing and system testing. Murphy *et al*. developed a supporting framework for conducting metamorphic testing at the function level [58]. They used the Java Modeling Language (JML) for specifying the metamorphic relations and automatically generating test code using the provided specifications. *Statistical Metamorphic testing (SMT)* is a technique for testing non-deterministic programs that lack oracles [25]. Guderlei *et al*. applied SMT for testing an implementation of the inverse cumulative distribution function of the normal distribution [25]. Further, SMT was applied for testing non-deterministic health care simulation software [57] and a stochastic optimization program [81].

Limitations: Enumerating a etof metamorphic relations that should be satisfied by a program is a critical initial task in applying metamorphic testing. A tester or developer has to manually identify metamorphic relations using her knowledge of the program under test; this manual process can miss some important metamorphic relations that could reveal faults. Recently we proposed a novel technique based on machine learning for automatically detecting metamorphic relations [34].

As noted in Section 3.2, selecting suitable tolerances for oracles is another challenge. Kelly *et al*. experimentally found that reducing the tolerance in an oracle increases the ability to detect faults in the code [36]. Clune *et al*. found that breaking the algorithm into small steps and testing the steps independently reduced the compounding effects of truncation and round-off errors [12].

Test case creation and selection—Several methods can help to overcome the challenges in test case creation and selection:

1. Hook *et al*. found that many faults can be identified by a small number of test cases that push the boundaries of the computation represented by the code [32]. Following this, Kelly *et al*. found that random tests combined with specially designed test cases to cover the parts of code uncovered by the random tests are effective in identifying faults [36]. Both of these studies used MATLAB functions in their experiments.
2. Randomly generated test cases were used with metamorphic testing to automate the testing of image processing applications [48].
3. Vilkomir *et al*. developed a method for automatically generating test cases when a scientific program has many input parameters with dependencies [76]. Vilkomir *et al*. represent the input space as a directed graph. Input parameters are represented by the nodes in the graph. Specific values of the parameters and the probability of a parameter taking that value are represented by arcs. Dependencies among input parameter values are handled by splitting/merging nodes. This method creates a model which satisfies the probability law of Markov chains. Valid test cases can be automatically generated by taking a path in this directed graph. This model also

provides the ability to generate random and weighted test cases according to the likelihood of taking the parameter values.

4. Bagnara *et al*. used symbolic execution to generate test data for floating point programs [5]. This method generates test data to traverse program paths that involve floating point computations.
5. Meinke *et al*. developed a technique for Automatic test case generation for numerical software based on learning based testing (LBT) [50]. The authors first createda polynomial model as an abstraction of the program under test. Then the test cases are generated by applying a satisfiability algorithm to the learned model.
6. Parameterized random data generation is a technique described by Murphy *et al*. [55] for creating test data for machine learning applications. This method randomly generates data sets using properties of equivalence classes.
7. Remmel *et al*. developed a regression testing framework for a complex scientific framework [63]. They took a software product line engineering (SPLE) approach to handle the large variability of the scientific framework. They developed a variability model to represent this variability and used the model to derive test cases while making sure necessary variant combinations are covered. This approach requires that scientists help to identify infeasible combinations.

Test coverage information—Only two primary studies mention the use of some type of test coverage information [33, 2]. Kane *et al*. found that while some developers were interested in measuring statement coverage, most of the developers were interested in covering the significant functionality of the program [33]. Ackroyd *et al*. [2] used the Emma tool to measure test coverage.

Assertion checking—Assertion checking can be used to ensure the correctness of plug-and-play scientific components. But assertion checking introduces a performance overhead. Dahlgren *et al*. developed an assertion selection system to reduce performance overhead for scientific software [15, 14].

Software development process—Several studies reported that using agile practices for developing scientific software improved testing activities [73, 61, 79]. Some projects have used test-driven development (TDD), where test are written to check the functionality before the code is written. But adopting this approach could be a cultural challenge since primary studies report that TDD can delay the initial development of functional code [29, 2].

3.4. RQ4: Are there any challenges that could not be answered by existing techniques?

Only one primary paper directly provided answers to RQ4. Kelly *et al*. [39] describes oracle problems as key problems to solve and the need for research on performing effective testing without oracles. We did not find other answers to this research question.

4. Discussion

4.1. Principal findings

The goal of this systematic literature review is to identify specific challenges faced when testing scientific software, how the challenges have been met, and any unsolved challenges. The principal findings of this review are the following:

1. The main challenges in testing scientific software can be grouped into two main categories.
    - Testing challenges that occur due to characteristics of scientific software.
        - Challenges concerning test case development such as a lack of real world data and difficulties in replicating the physical context where the scientific code is suppose to work.
        - Oracle problems mainly arise because scientific programs are either written to find answers that are previously unknown or they perform complex calculations so that it is difficult to determine the correct output. 30% of the primary studies reported the oracle problems as challenges for conducting testing.
        - Challenges towards test execution such as difficulties in running test cases to satisfy a coverage criteria due to long execution times.
        - Challenges towards test result interpretation such as round-off errors, truncation errors and model simplifications masking faults in the code.
    - Testing challenges that occur due to cultural differences between scientists and the software engineering community.
        - Challenges due to limited understanding of testing concepts such as viewing the code and the model that it implements as inseparable entities.
        - Challenges due to limited understanding of testing processes resulting in the use of ad-hoc or unsystematic testing methods.
        - Challenges due to not applying known testing methods such as unit testing.
2. We discovered how certain techniques can be used to overcome some of the testing challenges posed by scientific software development.
    - Pseudo oracles, analytical solutions, experimental results, measurement values, simplified data and professional judgment are widely used as solutions to oracle problems in scientific software. But we found no

empirical studies evaluating the effectiveness of these techniques in detecting subtle faults. New techniques such as metamorphic testing have been applied and evaluated for testing scientific software in research studies. But we found no evidence that such techniques are actually used in practice.

- Traditional techniques such as random test case generation were applied to test scientific software after applying modifications to consider equivalence classes. In addition, studies report the use of specific techniques to perform automatic test case generation for floating point programs. These techniques were only applied to a narrow set of programs. The applicability of these techniques in practice needs to be investigated.
- When considering unit, system, integration, acceptance and regression testing, very few studies applied more than one type of testing to their programs. We found no studies that applied more than three of these testing techniques.
- Only two primary studies evaluated some type of test coverage information during the testing process.

3. Research from the software engineering community can help to improve the testing process, by investigating how to perform effective testing for programs with oracle problems.

4.2. Techniques potentially useful in scientific software testing

Oracle problems are key problems to solve. Research on performing effective testing without oracles is needed [39]. Techniques such as property based testing and data redundancy can be used when an oracle is not available [4]. Assertions can be used to perform property based testing within the source code [35]. Another potential approach is to use a *golden run* [46]. With a golden run, an execution trace is generated during a failure free execution of an input. Then this execution trace is compared with execution traces obtained when executing the program with the same input when a failure is observed. By comparing the golden run and the faulty execution traces the robustness of the program is determined. One may also apply model based testing, but model based testing requires well-defined and stable requirements to develop the model. But with most scientific software, requirements are constantly changing, which can make it difficult to apply model based testing. We did not find applications of property based testing, data redundancy, golden run, and model based testing to test scientific software in the primary studies. In addition, research on test case selection and test data adequacy has not considered the effect of the oracle used. Often perfect oracles are not available for scientific programs. Therefore developing test selection/creation techniques that consider the characteristics of the oracle used for testing will be useful.

Metamorphic testing is a promising testing technique to address the oracle problem. Metamorphic testing can be used to perform both unit and system testing. But identifying metamorphic relations that should be satisfied by a program is challenging. Therefore techniques that can identify metamorphic relations for a program are needed [34].

Only a few studies applied new techniques developed by the software engineering community to overcome some of the common testing challenges. For example none of the primary studies employ test selection techniques to select test cases, even though running a large number of test cases is difficult due to the long execution times of scientific software. But many test selection techniques assume a perfect oracle, and thus will not work well for most scientific programs.

Several studies report that scientific software developers used regression testing during the development process. But we could not determine if regression testing was automated or whether any test case prioritizing techniques were used. In addition we only found two studies that used unit testing frameworks to conduct unit testing. Both of these studies report the use of the JUnit framework for Java programs. None of the primary studies report information regarding how unit testing was conducted for programs written in other languages.

One of the challenges of testing scientific programs is duplicated code. Even though a fault is fixed in a single location, the same fault may exist in other locations and those faults can go undetected when duplicated code is present. Automatic clone detection techniques would be useful to find duplicated code especially when dealing with legacy code.

4.3. Strengths and weaknesses of the SLR

Primary studies that provided the relevant information for this literature review were identified thorough a key word based search on three databases. The search found relevant studies published in journals, conference proceedings, and technical reports. We used a systematic approach, including the detailed inclusion/exclusion criteria given in Table 1 to select the relevant primary studies. Initially both authors applied the study selection process to a subset of the results returned by the key word based search. After verifying that both authors selected the same set of studies, the first author applied the study selection process to the rest of the results returned by the key word based search.

In addition we examined the reference lists of the selected primary studies to identify any additional studies that relate to our search focus. We found 13 additional studies related to our search focus. These studies were found by the key word based search, but did not pass the title based filtering. This indicates that selecting studies based on the title alone may not be reliable and to improve the reliability we might have to review the abstract, key words and conclusions before excluding them. This process would be time consuming due to the large number of results returned by the key word based search. After selecting the primary studies, we used data extraction forms to extract the relevant information consistently while reducing bias. Extracted information was validated by both authors.

We used the quality assessment questions given in Table 2 and Table 3 for assessing the quality of the selected primary studies. All selected primary studies are of high quality. The primary studies are a mixture of observational and experimental studies.

One weakness is the reliance on the key word based search facilities provided by the three databases for selecting the initial set of papers. We cannot ensure that the search facilities return all relevant studies. But, the search process independently returned all the studies that we previously knew as relevant to our research questions.

Many primary studies were published in venues that are not related to software engineering. Therefore, there may be solutions provided by the software engineering community for some of the challenges presented in Section 3.2 such as oracle problems. But we did not find evidence of wide use of such solutions by the scientific software developers.

4.4. Contribution to research and practice community

To our knowledge, this is the first systematic literature review conducted to identify software testing challenges, proposed solutions, and unsolved problems in scientific software testing. We identified challenges in testing scientific software using a large number of studies. We outlined the solutions used by the practitioners to overcome those challenges as well as unique solutions that were proposed to overcome specific problems. In addition we identified several unsolved problems.

Our work may contribute to focusing research efforts aiming at the improvement of testing of scientific software. This SLR will help the scientists who are developing software to identify specific testing challenges and potential solutions to overcome those challenges. In addition scientist developers can become aware of their cultural differences with the software engineering community that can impact software testing. Information provided here will help scientific software developers to carry out systematic testing and thereby improve the quality of scientific software. Further, there are many opportunities for software engineering research to find solutions to solve the challenges identified by this systematic literature review.

5. Conclusion and future work

Conducting testing to identify faults in the code is an important task in scientific software development that has received little attention. In this paper we present the results of a systematic literature review that identifies specific challenges faced when testing scientific software, how the challenges have been met and any unsolved challenges.. Below we summarize the answers to our four research questions

RQ1: How is scientific software defined in literature? Scientific software is defined as software with a large computational component. Further, scientific software is usually developed by multidisciplinary teams made up of scientists and software developers.

RQ2: Are there special characteristics or faults in scientific software or its devel- opment that make testing difficult? We identified two categories of challenges in scientific software testing: (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 model as inseparable entities.

RQ3: Can we use existing testing methods (or adapt them) to test scientific software effectively? A number of studies report on testing at different levels of abstraction such as unit testing, system testing and integration testing in scientific software development. Few studies report the use of unit testing frameworks. Many studies report the use of a pseudo oracle or experimental results to alleviate the lack of an oracle. In addition, several case studies report using metamorphic testing to test programs that do not have oracles. Several studies developed techniques to overcome challenges in test case creation. These techniques include the combination of randomly generated test cases with specially designed test cases, generating test cases by considering dependencies among input parameters, and using symbolic execution to generate test data for floating point programs. Only two studies use test coverage information.

RQ4: Are there challenges that could not be met by existing techniques? Oracle problems are prevalent and need further attention.

Scientific software poses special challenges for testing. Some of these challenges can be overcome by applying testing techniques commonly used by software engineers. Scientist developers need to incorporate these testing techniques into their software development process. Some of the challenges are unique due to characteristics of scientific software, such as oracle problems. Software engineers need to consider these special challenges when developing testing techniques for scientific software.

Acknowledgments

This project is supported by Award Number 1R01GM096192 from the National Institute of General Medical Sciences. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institute of General Medical Sciences or the National Institutes of Health. We thank the reviewers for their insightful comments on earlier versions of this paper.

References

1. Abackerli AJ, Pereira PH, Calonego N Jr. A case study on testing CMM uncertainty simulation software (VCMM). Journal of the Brazilian Society of Mechanical Sciences and Engineering. 2010 Mar.32:8–14.
2. Ackroyd K, Kinder S, Mant G, Miller M, Ramsdale C, Stephenson P. Scientific software development at a research facility. Software, IEEE. 2008 Jul-Aug;25(4):44–51.
3. Afzal W, Torkar R, Feldt R. A systematic review of search-based testing for nonfunctional system properties. Information and Software Technology. 2009; 51(6):957–976.
4. Ammann, P.; Offutt, J. Introduction to Software Testing, 1st Edition. New York, NY, USA: Cambridge University Press; 2008.
5. Bagnara, R.; Carlier, M.; Gori, R.; Gotlieb, A. Symbolic path-oriented test data generation for floating-point programs; Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on; 2013. p. 1-10.
6. Brilliant S, Knight J, Leveson N. Analysis of faults in an n-version software experiment. Software Engineering, IEEE Transactions on. 1990; 16(2):238–247.
7. Carver, J.; Kendall, RP.; Squires, SE.; Post, DE. Software development environments for scientific and engineering software: A series of case studies. Proceedings of the 29th International Conference on Software Engineering. ICSE ‘07. IEEE Computer Society; Washington, DC, USA. 2007. p. 550-559.

8. Carver, Jeffrey, RBDH.; Hochstein, L. Tech. Rep. SAND2011-2196. Sandia National Laboratories; 2011. What scientists and engineers think they know about software engineering: A survey.

9. Chen, T.; Feng, J.; Tse, TH. Metamorphic testing of programs on partial differential equations: a case study; Computer Software and Applications Conference, 2002. COMP-SAC 2002. Proceedings. 26th Annual International; 2002. p. 327-333.

10. Chen, TY.; Cheung, SC.; Yiu, SM. Tech. Rep. HKUST-CS98-01. Hong Kong: Department of Computer Science, Hong Kong University of Science and Technology; 1998. Metamorphic testing: a new approach for generating next test cases.

11. Chen TY, Ho JWK, Liu H, Xie X. An innovative approach for testing bioin-formatics programs using metamorphic testing. BMC Bioinformatics. 2009; 10

12. Clune T, Rood R. Software testing and verification in climate model development. Software, IEEE. 2011 Nov-Dec;28(6):49–55.

13. Cox M, Harris P. Design and use of reference data sets for testing scientific software. Analytica Chimica Acta. 1999; 380(23):339–351.

14. Dahlgren, T. Performance-driven interface contract enforcement for scientific components. Schmidt, H.; Crnkovic, I.; Heineman, G.; Stafford, J., editors. Springer Berlin Heidelberg: Component-Based Software Engineering. Vol. 4608 of Lecture Notes in Computer Science; 2007. p. 157-172.

15. Dahlgren, TL.; Devanbu, PT. Improving scientific software component quality through assertions; Proceedings of the Second International Workshop on Software Engineering for High Performance Computing System Applications. SE-HPCS ‘05. ACM; New York, NY, USA. 2005. p. 73-77.

16. Davis, MD.; Weyuker, EJ. Pseudo-oracles for non-testable programs; Proceedings of the ACM ‘81 conference. ACM ‘81. ACM; New York, NY, USA. 1981. p. 254-257.

17. Drake JB, Jones PW, Carr GR Jr. Overview of the software design of the community climate system model. International Journal of High Performance Computing Applications. 2005 Aug; 19(3):177–186.

18. Dubois P. Testing scientific programs. Computing in Science & Engineering. 2012 Jul-Aug;14(4): 69–73.

19. Dyba T, Dingsoyr T, Hanssen G. Applying systematic reviews to diverse study types: An experience report. Empirical Software Engineering and Measurement, 2007. 2007 Sep.:225–234. ESEM 2007. First International Symposium on.

20. Easterbrook, SM. Climate change: a grand software challenge; Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research. FoSER ‘10. ACM; New York, NY, USA. 2010. p. 99-104.

21. Easterbrook SM, Johns TC. Engineering the software for understanding climate change. Computing in Science & Engineering. 2009 Nov-Dec;11(6):65–74.

22. Eddins SL. Automated software testing for matlab. Computing in Science & Engineering. 2009; 11(6):48–55.

23. Engstrom E, Runeson P, Skoglund M. A systematic review on regression test selection techniques. Information and Software Technology. 2010; 52(1):14–30.

24. Farrell PE, Piggott MD, Gorman GJ, Ham DA, Wilson CR, Bond TM. Automated continuous verification for numerical simulation. Geoscientific Model Development. 2011; 4(2):435–449.

25. Guderlei, R.; Mayer, J. Statistical metamorphic testing testing programs with random output by means of statistical hypothesis tests and metamorphic testing; Quality Software, 2007. QSIC ‘07. Seventh International Conference on; 2007 Oct. p. 404-409.

26. Hannay, JE.; MacLeod, C.; Singer, J.; Langtangen, HP.; Pfahl, D.; Wilson, G. How do scientists develop and use scientific software?; Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering. SECSE ‘09. IEEE Computer Society; Washington, DC, USA. 2009. p. 1-8.

27. Hatton L. The T experiments: errors in scientific software. Computational Science & Engineering, IEEE. 1997 Apr-Jun;4(2):27–38.

28. Hatton L, Roberts A. How accurate is scientific software? Software Engineering, IEEE Transactions on. 1994 Oct; 20(10):785–797.

29. Heroux, M.; Willenbring, J. Barely sufficient software engineering: 10 practices to improve your CSE software; Software Engineering for Computational Science and Engineering, 2009. SECSE '09. ICSE Workshop on; 2009. p. 15-21.

30. Heroux, MA.; Willenbring, JM.; Phenow, MN. Improving the development process for CSE software; Parallel, Distributed and Network-Based Processing, 2007. PDP '07. 15th EUROMICRO International Conference on; 2007 Feb. p. 11-17.

31. Hochstein L, Basili V. The asc-alliance projects: A case study of large-scale parallel scientific code development. Computer. Mar; 41(3):50–58.

32. Hook, D.; Kelly, D. Testing for trustworthiness in scientific software; Software Engineering for Computational Science and Engineering, 2009. SECSE '09. ICSE Workshop on; 2009 May. p. 59-64.

33. Kane DW, Hohman MM, Cerami EG, McCormick MW, Kuhlmman KF, Byrd JA. Agile methods in biomedical software development: a multi-site experience report. BMC Bioinformatics. 2006; 7:273. [PubMed: 16734914]

34. Kanewala, U.; Bieman, J. Using machine learning techniques to detect metamorphic relations for programs without test oracles; Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on; 2013 Nov. p. 1-10.

35. Kanewala, U.; Bieman, JM. Techniques for testing scientific programs without an oracle; Proc. 5th International Workshop on Software Engineering for Computational Science and Engineering. IEEE; 2013. p. 48-57.

36. Kelly D, Gray R, Shao Y. Examining random and designed tests to detect code mistakes in scientific software. Journal of Computational Science. 2011; 2(1):47–56.

37. Kelly D, Hook D, Sanders R. Five recommended practices for computational scientists who write software. Computing in Science & Engineering. 2009 Sep-Oct;11(5):48–53.

38. Kelly, D.; Sanders, R. Assessing the quality of scientific software; First International Workshop on Software Engineering for Computational Science and Engineering; 2008.

39. Kelly D, Smith S, Meng N. Software engineering for scientists. Computing in Science & Engineering. 2011 Sep-Oct;13(5):7–11.

40. Kelly D, Thorsteinson S, Hook D. Scientific software testing: Analysis with four dimensions. Software, IEEE. 2011 May-Jun;28(3):84–90.

41. Kitchenham B. Procedures for performing systematic reviews. Technical report, Keele University and NICTA. 2004

42. Kitchenham B, Brereton OP, Budgen D, Turner M, Bailey J, Linkman S. Systematic literature reviews in software engineering a systematic literature review. Information and Software Technology. 2009; 51(1):7–15.

43. Kitchenham, B.; Charters, S. Technical report. Keele University and University of Durham; 2007. Guidelines for performing systematic literature reviews Vin software engineering (version 2.3).

44. Kreyman, K.; Parnas, DL.; Qiao, S. CRL Report no. 368. McMaster University; 1999. Inspection procedures for critical programs that model physical phenomena.

45. Lane PC, Gobet F. A theory-driven testing methodology for developing scientific software. Journal of Experimental & Theoretical Artificial Intelligence. 2012; 24(4):421–456.

46. Lemos, G.; Martins, E. Specification-guided golden run for analysis of robustness testing results; Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on; 2012 Jun. p. 157-166.

47. L.S. Chin DW, Greenough C. A survey of software testing tools for computational science. Tech. Rep. RAL-TR-2007-010, Rutherford Appleton Laboratory. 2007

48. Mayer, J.; Guderlei, R. On random testing of image processing applications; Quality Software, 2006. QSIC 2006. Sixth International Conference on; 2006 Oct. p. 85-92.

49. Mayer, J.; Informationsverarbeitung, AA.; Ulm, U. On testing image processing applications with statistical methods; In Software Engineering (SE 2005), Lecture Notes in Informatics; 2005. p. 69-78.

50. Meinke, K.; Niu, F. A learning-based approach to unit testing of numerical software. In: Petrenko, A.; Simo, A.; Maldonado, J., editors. Testing Software and Systems. Vol. 6435 of Lecture Notes in Computer Science. Springer Berlin Heidelberg; 2010. p. 221-235.

51. Miller G. A scientist’s nightmare: Software problem leads to five retractions. Science. 2006; 314(5807):1856–1857. [PubMed: 17185570]

52. Morris, C. Some lessons learned reviewing scientific code. Proc; First International Workshop on Software Engineering for Computational Science and Engineering; 2008.

53. Morris, C.; Segal, J. Some challenges facing scientific software developers: The case of molecular biology; e-Science, 2009. e-Science ‘09. Fifth IEEE International Conference on; 2009 Dec. p. 216-222.

54. Murphy, C.; Kaiser, G.; Arias, M. An approach to software testing of machine learning applications; 19th International Conference on Software Engineering and Knowledge Engineering (SEKE); 2007. p. 167-172.

55. Murphy, C.; Kaiser, G.; Arias, M. Parameterizing random test data according to equivalence classes; Proceedings of the 2nd international workshop on Random testing: co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007). RT ‘07. ACM; New York, NY, USA. 2007. p. 38-41.

56. Murphy, C.; Kaiser, G.; Hu, L.; Wu, L. Properties of machine learning applications for use in metamorphic testing; Proc. of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE); 2008 Jul. p. 867-872.

57. Murphy, C.; Raunak, MS.; King, A.; Chen, S.; Imbriano, C.; Kaiser, G.; Lee, I.; Sokolsky, O.; Clarke, L.; Osterweil, L. On effective testing of health care simulation software; Proceedings of the 3rd Workshop on Software Engineering in Health Care. SEHC ‘ 11. ACM; New York, NY, USA. 2011. p. 40-47.

58. Murphy, C.; Shen, K.; Kaiser, G. Using JML runtime assertion checking to automate metamorphic testing in applications without test oracles; Proceedings of the 2009 International Conference on Software Testing Verification and Validation. ICST ‘09. IEEE Computer Society; Washington, DC, USA. 2009. p. 436-445.

59. Nguyen-Hoan, L.; Flint, S.; Sankaranarayana, R. A survey of scientific software development; Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM ‘10. ACM; New York, NY, USA. 2010. p. 12:1-12:10.

60. Pipitone J, Easterbrook S. Assessing climate model software quality: a defect density analysis of three models. Geoscientific Model Development. 2012; 5(4):1009–1022.

61. Pitt-Francis J, Bernabeu MO, Cooper J, Garny A, Momtahan L, Osborne J, Path-manathan P, Rodriguez B, Whiteley JP, Gavaghan DJ. Chaste: using agile programming techniques to develop computational biology software. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences. 2008; 366(1878):3111–3136.

62. Post DE, Kendall RP. Software project management and quality engineering practices for complex, coupled multiphysics, massively parallel computational simulations: Lessons learned from ASCI. International Journal of High Performance Computing Applications. 2004 Winter;18(4):399–416.

63. Remmel H, Paech B, Bastian P, Engwer C. System testing a scientific framework using a regression-test environment. Computing in Science & Engineering. 2012 Mar-Apr;14(2):38–45.

64. Reupke, W.; Srinivasan, E.; Rigterink, P.; Card, D. The need for a rigorous development and testing methodology for medical software; Engineering of Computer-Based Medical Systems, 1988., Proceedings of the Symposium on the; 1988 Jun. p. 15-20.

65. Sanders, R.; Kelly, D. The challenge of testing scientific software; Proceedings Conference for the Association for Software Testing (CAST); 2008 Jul. p. 30-36.

66. Sanders R, Kelly D. Dealing with risk in scientific software development. Software, IEEE. 2008 Jul-Aug;25(4):21–28.

67. Segal J. When software engineers met research scientists: A case study. Empirical Software Engineering. 2005; 10:517–536.

68. Segal J. Some problems of professional end user developers. Visual Languages and Human-Centric Computing. 2007:111–118. 2007. VL/HCC 2007. IEEE Symposium on.

69. Segal, J. Models of scientific software development; 2008 Workshop Software Eng. in Computational Science and Eng. (SECSE 08); 2008.

70. Segal, J. In: Buckley, J.; VRooksby, J.; Bednarik, R., editors. Scientists and software engineers: A tale of two cultures; PPIG 2008: Proceedings of the 20th Annual Meeting of the Pschology of

Programming Interest Group. Lancaster University, Lancaster, UK, proceedings: 20th annual meeting of the Psychology of Programming Interest Group; Lancaster; United Kingdom. 2008 Sep. p. 10-12.2008

71. Segal J. Software development cultures and cooperation problems: a field study of the early stages of development of software for a scientific community. Computer Supported Cooperative Work. 2009 Dec; Winter;18(5–6):581–606.
72. Segal, J. Some challenges facing software engineers developing software for scientists; Software Engineering for Computational Science and Engineering, 2009. SECSE ‘09. ICSE Workshop on; 2009 May. p. 9-14.
73. Sletholt M, Hannay J, Pfahl D, Langtangen H. What do we know about scientific software development’s agile practices? Computing in Science & Engineering. 2012 Mar; 14(2):24–37.
74. Smith, B. In: Gaffney, P.; Pool, J., editors. A test harness th for numerical applications and libraries; Grid-Based Problem Solving Environments. Vol. 239 of IFIP The International Federation for Information Processing; 2007. p. 227-241.
75. Smith, MC.; Kelsey, RL.; Riese, JM.; Young, GA. In: Trevisani, DA.; Sisti, AF., editors. Creating a flexible environment for testing scientific software; Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. Vol. 5423 of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series; 2004 Aug. p. 288-296.
76. Vilkomir, SA.; Swain, WT.; Poore, JH.; Clarno, KT. Modeling input space for testing scientific computational software: A case study; Proceedings of the 8th international conference on Computational Science, Part III. ICCS ‘08; 2008. p. 291-300.
77. Walia GS, Carver JC. A systematic literature review to identify and classify software requirement errors. Information and Software Technology. 2009; 51(7):1087–1109.
78. Weyuker EJ. On testing non-testable programs. The Computer Journal. 1982; 25(4):465–470.
79. Wood W, Kleb W. Exploring xp for scientific research. Software, IEEE. 2003; 20(3):30–36.
80. Xie X, Ho JW, Murphy C, Kaiser G, Xu B, Chen TY. Testing and validating machine learning classifiers by metamorphic testing. Journal of Systems and Software. 2011; 84(4):544–558. [PubMed: 21532969]
81. Yoo, S. Metamorphic testing of stochastic optimisation; Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on; 2010. p. 192-201.

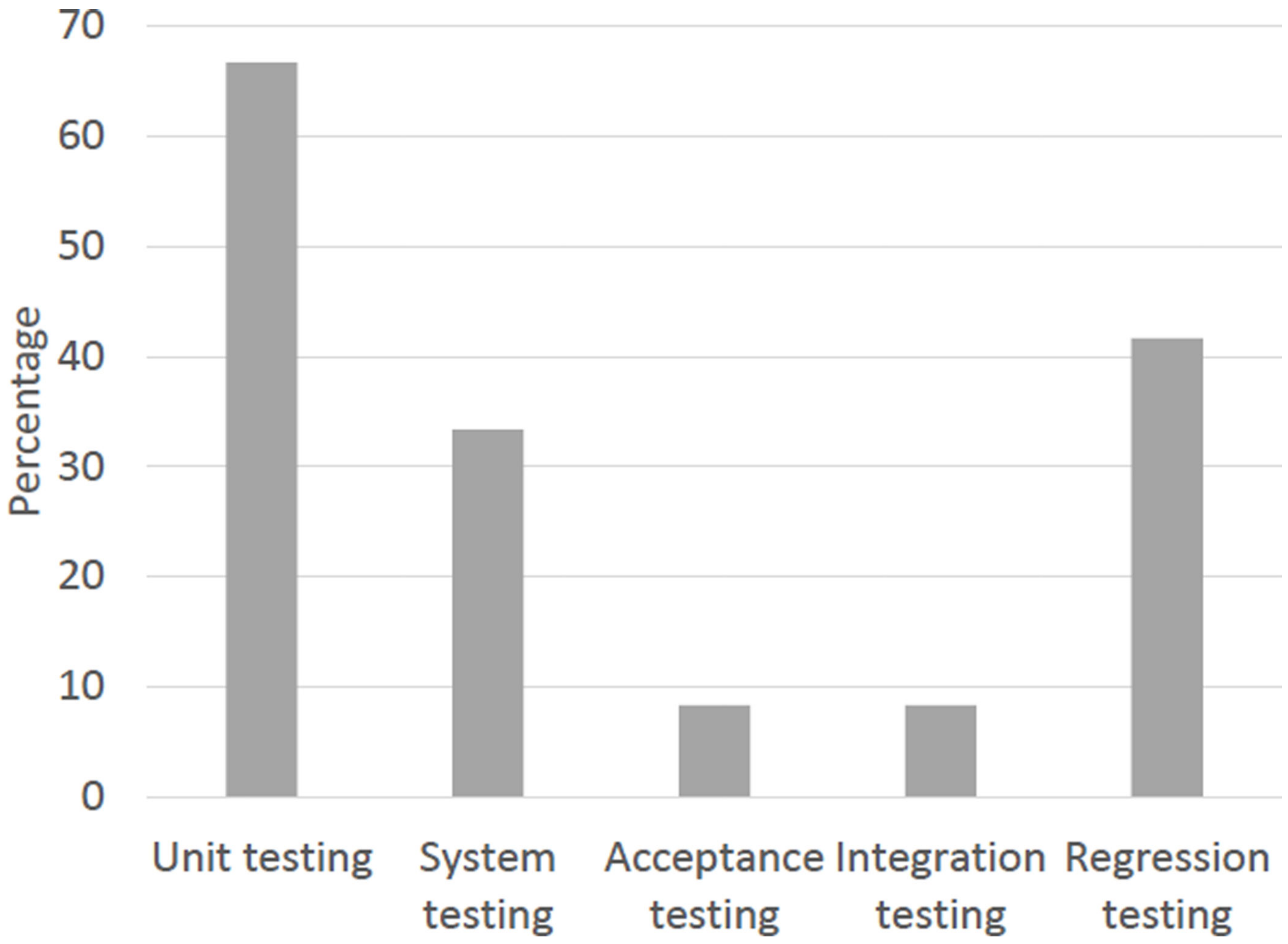

Figure 1.
Percentage of studies that applied different testing methods

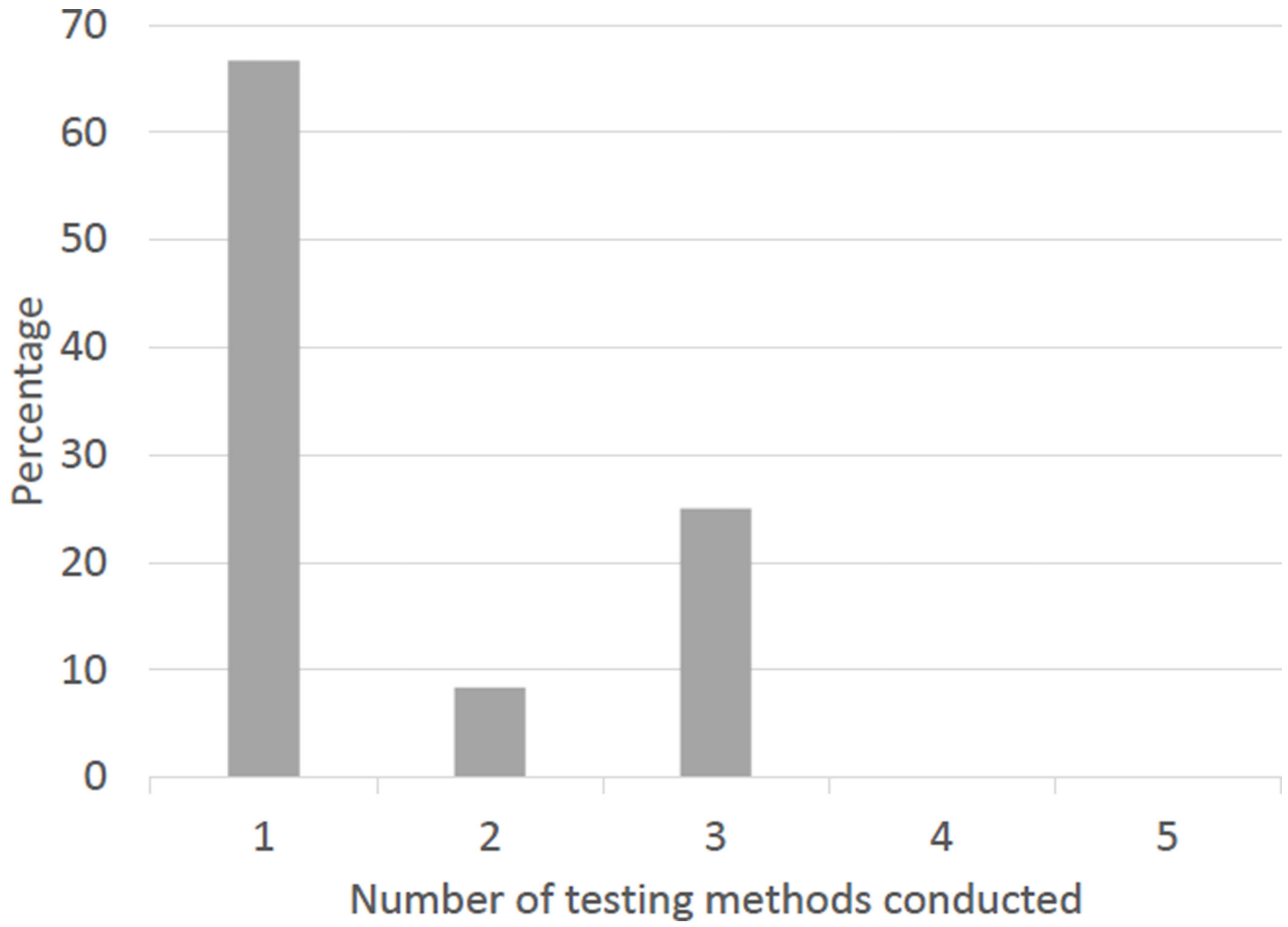

Figure 2.
Number of testing methods applied by the studies

Table 1

Inclusion and exclusion criteria

Inclusion criteria	**Exclusion criteria**
1 Papers that describe characteristics of scientific software that impact testing.	**1** Papers that present opinions without sufficient evidence supporting the opinion.
2 Case studies or surveys of scientific soft-ware testing experiences.	**2** Studies not related to the research questions.
3 Papers that analyze characteristics of scientific software testing including case studies and experience reports.	**3** Studies in languages other than English.
4 Papers describing commonly occurring faults in scientific software.	**4** Papers presenting results without providing supporting evidence.
5 Papers that describe testing methods used for scientific software and provide a sufficient evaluation of the method used.	**5** Preliminary conference papers of included journal papers.
6 Experience reports or case studies describing testing methods used for scientific software.	

Table 2

Quality assessment for quantitative studies

Survey	Case study	Experiment
G1: Are the study aims clearly stated?		
S1: Was the method for collecting the sample data specified (e.g. postal, interview, web-based)?	N/A	N/A
S2: Is there a control group?	N/A	E1: Is there a control group?
N/A	N/A	E2: Were the treatments randomly allocated?
G2: Are the data collection methods adequately described?		
G3: Was there any statistical assessment of results?		
S3: Do the observations support the claims?	C1: Is there enough evidence provided to support the claims?	E3: Is there enough evidence provided to support the claims?
G4: Are threats to validity and/or limitations reported?		
G5: Can the study be replicated?		

Table 3

Quality assessment for qualitative studies

Quality assessment questions	
1	A: Are the study aims clearly stated?
2	B: Does the evaluation address its stated aims and purpose?
3	C: Is sample design/target selection of cases/documents defined?
4	D: Is enough evidence provided to support the claims?
5	E: Can the study be replicated?

Table 4

Data extraction form

Search focus	Data Item	Description
General	Identifier	Reference number given to the article
	Bibliography	Author, year, Title, source
	Type of article	journal/conference/tech. report
	Study aims	Aims or goals of the study
	Study design	controlled experiment/survey/etc.
RQ1	Definition	Definition for scientific software
	Examples *A*	Examples of scientific software
RQ2	Challenge/problem	Challenges/problems faced when testing scientific soft-ware
	Fault description	Description of the fault found
	Causes	What caused the fault?
RQ3/RQ4	Testing method	Description of the method used
	Existing/new/extension	Whether the testing method is new, existing or modification to an existing method
	Challenge/problem	The problem/challenge that it provides the answer to
	Faults/failures found	Description of the faults/ failures found by the method
	Evidence	Evidence for the effectiveness of the method of finding faults
	Limitations	Limitations of the method

Table 5

Selected Primary Studies (Part 1)

Study No.	Ref. No.	Study focus	RQ1	RQ2	RQ3	RQ4
PS1	[1]	A case study on testing software packages used in metrology		✓	✓	
PS2	[2]	Software engineering tasks carried out during scientific software development		✓		
PS3	[5]	Test case generation for floating point programs using symbolic execution		✓		
PS4	[7]	Case studies of scientific software development projects		✓		
PS5	[8]	Survey on computational scientists and engineers	✓	✓		
PS6	[9]	Applying metamorphic testing to programs on partial differential equations		✓	✓	
PS7	[11]	Case studies on applying metamorphic testing for bioinformatics programs		✓	✓	
PS8	[12]	Case studies on applying test driven development for climate models		✓	✓	
PS9	[13]	Using reference data sets for testing scientific software			✓	
PS10	[14]	Effectiveness of different interface contract enforcement policies for scientific components			✓	
PS11	[15]	Partial enforcement of assertions for scientific software components			✓	
PS12	[16]	Using pseudo-oracles for testing programs without oracles			✓	
PS13	[17]	A case study on developing a climate system model			✓	
PS14	[18]	A tool for automating the testing of scientific simulations		✓		
PS15	[20]	Discussion on software challenges faced in climate modeling program development		✓		
PS16	[21]	Ethnographic study of climate scientists who develop software		✓	✓	
PS17	[22]	A unit testing framework for MATLAB programs		✓	✓	
PS18	[24]	A framework for automated continuous verification of numerical simulations			✓	
PS19	[26]	Results of a survey conducted to identify how scientists develop and use software in their research		✓		
PS20	[27]	Experiments to analyze the accuracy of scientific software through static analysis and comparisons with independent implementations of the same algorithm		✓	✓	
PS21	[28]	N-version programming experiment conducted on scientific software		✓		
PS22	[30]	Applying software quality assurance practices in a scientific software project		✓		
PS23	[29]	Software engineering practices suitable for scientific software development teams identified through a case study		✓	✓	
PS24	[31]	Software development process of five large scale computational science software			✓	
PS25	[32]	Evaluating the effectiveness of using a small number of carefully selected test cases for testing scientific software		✓	✓	
PS26	[33]	Qualitative study of agile development approaches for creating and maintaining bio-medical software			✓	

Study No.	Ref. No.	Study focus	RQ1	RQ2	RQ3	RQ4
PS27	[36]	Comparing the effectiveness of random test cases and designed test cases for detecting faults in scientific software		✓	✓	
PS28	[37]	Useful software engineering techniques for computational scientists obtained through experience of scientists who had success		✓		
PS29	[38]	Quality assessment practices of scientists that develop computational software		✓	✓	
PS30	[39]	How software engineering research can provide solutions to challenges found by scientists developing software	✓	✓		✓

Table 6

Selected Primary Studies (Part 2)

Study No.	Ref. No.	Study focus	RQ1	RQ2	RQ3	RQ4
PS31	[40]	A case study of applying testing activities to scientific software		✓	✓	
PS32	[47]	A survey on testing tools for scientific programs written in FOR- TRAN		✓		
PS33	[45]	A case study on using a three level testing architecture for testing scientific programs			✓	
PS34	[48]	Applying metamorphic testing for image processing programs			✓	
PS35	[49]	Using statistical oracles to test image processing applications			✓	
PS36	[50]	A learning-based method for automatic generation of test cases for numerical programs			✓	
PS37	[52]	Lessons learned through code reviews of scientific programs				
PS38	[53]	Challenges faced by software engineers developing software for scientists in the field of molecular biology	✓			
PS39	[55]	A framework for randomly generating large data sets for testing machine learning applications			✓	
PS40	[54]	Methods for testing machine learning algorithms		✓	✓	
PS41	[56]	Applying metamorphic testing for testing machine learning applications			✓	
PS42	[57]	Testing health care simulation software using metamorphic testing		✓	✓	
PS43	[59]	Survey of scientific software developers		✓	✓	
PS44	[60]	Analysis of quality of climate models in terms of defect density		✓		
PS45	[61]	Applying agile development process for developing computational biology software		✓	✓	
PS46	[62]	Lessons learned from scientific software development projects			✓	
PS47	[63]	Applying variability modeling for selecting test cases when testing scientific frameworks with large variability		✓	✓	
PS48	[64]	Medical software development and testing			✓	
PS49	[66]	A survey to identify the characteristics of scientific software development		✓	✓	
PS50	[65]	Challenges faced when testing scientific software identified through interviews carried out with scientists who develop/use scientific software	✓	✓	✓	
PS51	[70]	Study of problems arising when scientists and software engineers work together to develop scientific software		✓		
PS52	[68]	Problemsin scientific software development identified through case studies in different fields		✓		
PS53	[72]	Challenges faced by software engineers who develop software for scientists		✓		
PS54	[71]	Case studies on professional end-user development culture		✓		
PS55	[69]	A model of scientific software development identified through multiple field studies of scientific software development		✓		
PS56 ,	[67]	A case study on applying traditional document-led development methodology for developing scientific software		✓		

Study No.	Ref. No.	Study focus	RQ1	RQ2	RQ3	RQ4
PS57	[73]	Literature review and case studies on how scientific software development matches agile practices and the effects of using agile practices in scientific software development		✓	✓	
PS58	[74]	A test harness for numerical programs			✓	
PS59	[75]	A testing framework for conducting regression testing of scientific software		✓	✓	
PS60	[76]	A method for test case generation of scientific software when there are dependencies between input parameters		✓	✓	
PS61	[78]	Testing non-testable programs		✓	✓	
PS62	[79]	Culture clash when applying extreme programming to develop scientific software		✓	✓	

Table 7

Publication venues of primary studies

Publication venue	Type	Count	%
International Workshop on Software Engineering for Computational Science and Engineering	Workshop	7	11.3
Computing in Science & Engineering	Journal	7	11.3
IEEE Software	Journal	5	8.1
BMC Bioinformatics	Journal	2	3.2
Geoscientific Model Development	Journal	2	3.2
International Conference on Software Engineering and Knowledge Engineering	Conference	2	3.2
International Journal of High Performance Computing Applications	Journal	2	3.2
Lecture Notes in Computer Science	Book chapter	2	3.2
Journal of the Brazilian Society of Mechanical Sciences and Engineering	Journal	1	1.6
International Conference on Software Testing, Verification and Validation	Conference	1	1.6
International Conference on Software Engineering	Conference	1	1.6
Sandia National Laboratories-Technical report	Tech. report	1	1.6
Computer Software and Applications Conference	Conference	1	1.6
Analytica Chimica Acta	Journal	1	1.6
International Workshop on Software Engineering for High Performance Computing System Applications	Workshop	1	1.6
ACM ‘81 Conference	Conference	1	1.6
FSE/SDP Workshop on Future of Software Engineering Research	Workshop	1	1.6
IEEE Computational Science & Engineering	Journal	1	1.6
IEEE Transactions on Software Engineering	Journal	1	1.6
EUROMICRO International Conference on Parallel, Distributed and Network- Based Processing	Conference	1	1.6
IEEE Computer	Journal	1	1.6
Journal of Computational Science	Journal	1	1.6
Rutherford Appleton Laboratory-Technical report	Tech. report	1	1.6
Journal of Experimental & Theoretical Artificial Intelligence	Journal	1	1.6
International Conference on Quality Software	Conference	1	1.6
Lecture Notes in Informatics	Book chapter	1	1.6
International Conference on e-Science	Conference	1	1.6
International Workshop on Random testing	Conference	1	1.6
Workshop on Software Engineering in Health Care	Workshop	1	1.6
International Symposium on Empirical Software Engineering and Measurement	Conference	1	1.6
Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences	Journal	1	1.6
Symposium on the Engineering of Computer-Based Medical Systems	Conference	1	1.6
Conference for the Association for Software Testing	Conference	1	1.6
Annual Meeting of the Psychology of Programming Interest Group	Conference	1	1.6
Symposium on Visual Languages and Human-Centric Computing	Conference	1	1.6
Computer Supported Cooperative Work	Journal	1	1.6
Empirical Software Engineering	Journal	1	1.6

Publication venue	**Type**	**Count**	**%**
Grid-Based Problem Solving Environments	Journal	1	1.6
Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series	Conference	1	1.6
International Conference on Computational Science	Conference	1	1.6
The Computer Journal	Journal	1	1.6

Table 8

Quality assessment results of quantitative studies

Ref. No.	G1	S1	S2	E1	E2	G2	G3	S3	C1	E3	G4	G5
[17]	yes	N/A	N/A	N/A	N/A	no	no	N/A	yes	N/A	no	no
[57]	yes	N/A	N/A	no	no	yes	no	N/A	N/A	yes	yes	yes
[55]	yes	N/A	N/A	no	no	yes	no	N/A	N/A	yes	yes	yes
[36]	yes	N/A	N/A	yes	no	yes	yes	N/A	N/A	yes	no	yes
[60]	yes	N/A	N/A	N/A	N/A	yes	no	N/A	yes	N/A	yes	yes
[1]	yes	N/A	N/A	N/A	N/A	yes	no	N/A	yes	N/A	no	yes
[13]	yes	N/A	N/A	no	no	yes	yes	N/A	N/A	yes	no	yes
[15]	yes	N/A	N/A	yes	no	yes	no	N/A	N/A	yes	no	yes
[8]	yes	yes	no	N/A	N/A	yes	yes	yes	N/A	N/A	no	yes
[59]	yes	yes	no	N/A	N/A	yes	no	yes	N/A	N/A	yes	yes
[61]	yes	N/A	N/A	N/A	N/A	yes	no	N/A	yes	N/A	no	yes
[63]	yes	N/A	N/A	N/A	N/A	yes	no	N/A	yes	N/A	no	yes
[32]	yes	N/A	N/A	yes	no	yes	no	N/A	N/A	yes	no	yes
[14]	yes	N/A	N/A	yes	no	yes	no	N/A	N/A	yes	no	yes
[48]	yes	N/A	N/A	no	no	yes	no	N/A	N/A	yes	no	yes
[11]	yes	N/A	N/A	N/A	N/A	yes	no	N/A	yes	N/A	yes	yes
[9]	yes	N/A	N/A	N/A	N/A	yes	no	N/A	yes	N/A	no	yes
[79]	yes	N/A	N/A	N/A	N/A	yes	no	N/A	yes	N/A	yes	no
[5]	yes	N/A	N/A	yes	no	yes	no	N/A	N/A	yes	no	yes
[26]	yes	yes	no	N/A	N/A	yes	no	yes	N/A	N/A	yes	yes
[27]	yes	N/A	N/A	no	no	yes	yes	N/A	N/A	yes	no	no
[28]	yes	N/A	N/A	no	no	yes	yes	N/A	N/A	yes	no	no
[50]	yes	N/A	N/A	yes	no	yes	no	N/A	N/A	yes	no	yes
[79]	yes	N/A	N/A	N/A	N/A	yes	no	N/A	yes	N/A	no	yes

G1: Are the study aims clearly stated?

S1: Was the method for collecting the sample data specified?

S2, E1: Is there a control group?

E2: Were the treatments randomly allocated?

G2: Are the data collection methods adequately described?

G3: Was there any statistical assessment of results?

S3: Do the observations support the claims?

C1, E3: Is there enough evidence provided to support the claims?

G4: Are threats to validity and/or limitations reported?

G5: Can the study be replicated?

Table 9

Quality assessment results of qualitative studies

Ref. No.	A	B	C	D	E
[62]	yes	yes	yes	yes	yes
[64]	yes	yes	no	yes	no
[67]	yes	yes	no	yes	no
[53]	yes	yes	yes	yes	no
[2]	yes	yes	yes	yes	no
[40]	yes	yes	no	yes	no
[39]	yes	yes	yes	yes	no
[75]	yes	yes	no	yes	no
[30]	yes	yes	yes	yes	no
[47]	yes	yes	no	yes	no
[20]	yes	yes	no	yes	no
[38]	yes	yes	yes	yes	no
[66]	yes	yes	yes	yes	no
[12]	yes	yes	no	yes	no
[18]	yes	yes	yes	yes	yes
[74]	yes	yes	yes	yes	no
[45]	yes	yes	yes	yes	no
[56]	yes	yes	yes	yes	yes
[31]	yes	yes	yes	yes	yes
[49]	yes	yes	yes	yes	yes
[16]	yes	yes	no	yes	no
[70]	yes	yes	yes	yes	yes
[71]	yes	yes	yes	yes	yes
[7]	yes	yes	yes	yes	no
[68]	yes	yes	yes	yes	no
[54]	yes	yes	yes	yes	yes
[29]	yes	yes	yes	yes	no

Ref. No.	**A**	**B**	**C**	**D**	**E**
[33]	yes	yes	yes	yes	no
[24]	yes	yes	yes	yes	no
[37]	yes	yes	no	yes	no
[52]	yes	yes	yes	yes	yes
[21]	yes	yes	yes	yes	yes
[22]	yes	yes	no	no	no
[65]	yes	yes	yes	yes	yes
[72]	yes	yes	yes	yes	no
[69]	yes	yes	yes	yes	no
[73]	yes	yes	yes	yes	yes
[78]	yes	yes	no	no	no

A: Are the study aims clearly stated?

B: Does the evaluation address its stated aims and purpose?

C: Is sample design/target selection of cases/documents defined?

D: Is enough evidence provided to support the claims?

E: Can the study be replicated?

Table 10

Details of scientific software listed in primary studies

Ref. No.	Description	Programing language	Size
[64]	Medical software (e.g. software for blood chemistry analyzer and medical image processing system)	N/S	N/S
[62]	Nuclear weapons simulation software	FORTRAN	500 KLOC
[17, 20]	Climate modeling software	N/S	N/S
[67]	Embedded software for spacecrafts	N/S	N/S
[53]	Software developed for space scientists and biologists	N/S	N/S
[2]	Control and data acquisition software for Synchrotron Radiation Source (SRS) experiment stations	Java	N/S
[57]	Health care simulation software(e.g. discreet event simulation engine and insulin titration algorithm simulation)	Java, MAT- LAB	N/S
[55]	Machine learning ranking algorithm implementations	Perl, C	N/S
[60]	Climate modeling software	FORTRAN, C	400 KLOC
[40]	Astronomy software package	MAT LAB, C++	10 KLOC
[1]	Software packages providing uncertainty estimates for tri-dimensional measurements	N/S	N/S
[75]	Implementation of a time dependent simulation of a complex physical system	N/S	N/S
[15]	Implementation of scientific mesh traversal algorithms	N/S	38–50 LOC
[30]	Implementations of parallel solver algorithms and libraries for large scale, complex, multi physics engineering and scientific applications	N/S	N/S
[61]	Software for cardiac modeling in computational biology	C++, Python	50 KLOC
[24]	Numerical simulations in geophysical fluid dynamics	N/S	N/S
[63]	Program for solving partial differential equations	C++	250 KLOC
[12]	Calculates the trajectory of the billions of air particles in the atmosphere	C++	N/S
[12]	Implementation of a numerical model that simulates the growth of virtual snow flakes	C++	N/S
[14]	Implementations of mesh traversal algorithms	N/S	N/S
[48]	Image processing application	N/S	N/S
[11]	Bioinformatics program for analyzing and simulating gene regulatory net- works and mapping short sequence reads to a reference genome	N/S	N/S
[55, 54]	Implementations of machine learning algorithms	N/S	N/S
[31]	Simulations in solid mechanics, fluid mechanics and combustion	C, C++, FOR- TRAN	100–500 KLOC
[79]	Program to evaluate the performance of a numerical scheme to solve a model advection-diffusion problem	Ruby	2.5 KLOC
[49]	Implementation of dilation of binary images	N/S	N/S
[71]	Infrastructure software for the structural protein community	N/S	N/S
[7]	Performance prediction software for a product that otherwise requires large, expensive and potentially dangerous empirical tests for performance evaluation	FORTRAN, C	405 KLOC
[7]	Provide computational predictions to analyze the manufacturing process of composite material products	C++, C	134 KLOC
[7]	Simulation of material behavior when placed under extreme stress	FORTRAN	200 KLOC
[7]	Provide real-time processing of sensor data	C++, MAT- LAB	100 KLOC

Ref. No.	Description	Programing language	Size
[7]	Calculate the properties of molecules using computational quantum mechanical models	FORTRAN	750 KLOC
[5]	Program for avoiding collisions in unmanned aircrafts	C	N/S
[29]	Numerical libraries to be used by computational science and engineering software projects	N/S	N/S