Software Engineering Practices and Principles to Increase Quality of Scientific Applications

Bojana Koteska and Anastas Mishev

Faculty of Computer Science and Engineering,
Rugjer Boshkovikj 16, P.O. Box 393 1000 Skopje, Macedonia
bojana.koteska@gmail.com,anastas.mishev@finki.ukim.mk

Abstract. The goal of this paper is to propose some software engineering practices and principles that could increase the quality of scientific applications. Since standard principles of software engineering cannot fully engage in enhancing the development process of such applications, finding the right principles and their combination that will improve the quality is real challenge for software engineers. In order to provide more realistic representation of problems in the field of scientific high-performance computing, we conducted a survey where developers of scientific applications in the HP-SEE project answered some key questions about testing methods and conventions they used. Analysis of the results of the responses was major indicator of quality deficiencies in the high-performance scientific software development and that helped us to discern the possible improvements that need to be added in the planning, development and particularly in the verification phases of the software life cycle.

Keywords: Scientific Applications Quality, Software Engineering Principles, HP-SEE project.

1 Introduction

Solving complex problems in the field of natural sciences, especially in the last decade, strongly implies the need for inclusion of software engineering as an area that can certainly make a significant contribution in terms of improving performance, organization and quality of scientific applications. Since scientific computing is a multidisciplinary field, the need for computer modeling of scientific processes in order to get quantitative results is inevitable [5]. Thus confirming that the design of software which simulate any scientific phenomena is crucial for science and it could provide great support when it comes to automation of certain parts of the development process such as data analysis, management of workflows [7], testing, etc.

Practices and principles of software engineering impose new views and perceptions when it comes to raising the quality of high-performance scientific applications. Software engineering can be defined as a formal set of procedures and tools ensuring efficient project development [14] that could be used and adapted to support the entire life cycle of scientific applications. The complexity of the domain which is the basic difference between the development of commercial and scientific applications [16] shows that the principles of software engineering cannot be fully applied in same way as in the development of commercial applications, so finding the appropriate combination and modification for their adjustment is a real challenge for software engineers.

Simulations of problems in science require maximum commitment and correct design that provide high processing performance and certainly better response time [8]. This shows that it is very important to pay attention to the applications quality and to make improvement efforts in order to create efficient development process and to detect and eliminate errors as soon as possible.

Hence, the main focus of this paper is aimed on identifying and proposing practices of software engineering that can significantly contribute to key improvements in the quality of scientific applications. The goal is, through research of already accepted quality recommendations, as well as practical understanding of the real problems faced by scientists, to define a quality model which will form the basis of what standards and criteria should be included in the quality review of scientific applications. Given the fact that standard principles of software engineering cannot fully apply in the development of scientific applications, the research involves the selection and adaptive combination of practices based on real answers from scientists who are part of ongoing development of this type of applications in the HP-SEE project. The results of detailed research helped to establish criteria and quality requirements whose application would significantly improve the planning and implementation practices for testing the quality of scientific applications.

2 Different Cultures of Software Engineers and Scientists

Seeing the potential problems that may occur as a result of different views of software engineers and scientists on the development process of scientific applications, especially from the vantage point of the quality of the applications, in particular, is the most important thing for this research. The obtained results will have a large impact in proposing recommendations that will remove some of the deficiencies and will solve future problems quickly.

One of the main reasons contributing to poor quality high performance computing applications is the lack of used software engineering formal methods. Since scientists have little or not knowledge of software engineering principles [16] and they learn programming techniques independently or by other scientists and not by software engineers [2], it is very difficult for them to adapt to the practices offered by software engineering.

Some scientists believe that the software development is only the process of coding and that it is not important for improving their scientific career because they think that many people can do it also [16]. They often create the code alone or they develop in small teams without previous formal training [14] [15] or they took the code from a source that is part of their research group [2], so the code refactoring and optimization process that will enable parallelism becomes even harder later. The first version of the code is usually created by scientists [15].

The first problem that software engineers could face when they are part of a scientific application development process is the meager list of requirements created by the scientists. That happens because creating requirements at the beginning is very difficult process due to the requirement changes, changes in models and algorithms changes [14] [15]. Software engineers expect a formal list of requirements, while scientists believe that many of the requirements should be discussed directly with engineers and that there is no need of written form [15]. Scientists do not use a formal approach for writing requirements and they do not create requirements document, so when performing software testing they do not know which requirements are completed and which are not. On the other hand, requirements could be written in high level language by using specific keywords in the scientific domain that software engineer cannot understand because he has no knowledge of the problem nature and science community practices [16][15].

Applications created by the scientists are mostly used in their research group and the most important thing for them is to get scientifically accurate result supported by well-simulated models [2] [15]. This clearly shows that they do not care how the processes of making calculations and optimizations are performed. Scientists are often guided by the thought that if a hypothesis is proven true, there is no longer need for reprogramming that part of the code [3]. The process of optimization becomes a matter of discussion among scientists only when they are unable to get results by using the available resources. If the code optimization is made, they want to use the free space to add new functionalities. They prefer readability of complex mathematical formulas rather than optimization which will be accepted only if necessary. It is very difficult to convince scientists that it is easier to use the existing frameworks, rather than creating new ones. Scientists do not accept changes in technology easy because they believe that making long term code and doing things from scratch are very difficult processes [2].

The processes of code verification and validation are differently accepted by scientists. They primarily give importance to the algorithm quality and much less to the implementation [2]. The verification process of high performance computing software is difficult [9] and it requires software engineers to be familiar with the scope of the problem because that science field is unknown for them [16]. However, providing the confidence that the solution is exactly what scientists in the community are looking to accomplish, is engineer's task [3].

Simulations of problems in the scientific world have no precise solution, but they are approximated with numerical algorithm. The process of validation usually means comparison with experimental results that sometimes are impossible to be obtained, so the problem of verification and validation is always present [14]. It's almost impossible to achieve 100% reliability because software engineers can not fully understand the theory and scientists the code [1]. Some errors could be found within the software code, but errors in the implementation logic could occur also [12]. Scientists validate software according to whether there is some progress in scientific research, so they are not very interested in the validation process until an error that affects scientific results is not found [16]. Therefore the creation of tests for scientific applications is a real challenge for software engineers.

3 Background Work

Software engineering can greatly affect the quality and productivity of scientific software in two ways, through introduction of new engineering practices that have proved as good in large projects by monitoring and publication of those already adapted [2]. The application of software engineering principles in the development process of high performance scientific software, can basically be the same with the application of software engineering in the development process of any application, but it requires only small additional changes in terms of planning, eliciting requirements, testing and development process [3]. Including software engineering practices and recommendations in the development of scientific applications requires making careful choices because their improper combination can decrease the quality of the applications which is contrary to our expectations.

The standard definitions of verification and validation cannot be fully applied in the scientific software development due to the facts that process of defining requirements is difficult and no external users exist. In the scientific world, verification can be defined as a process that ensures that the equations are solved properly without coding errors, i.e. in some way it is connected with mathematical models. Validation ensures that the results obtained from the code are accurate reproduction of the problem and that models that simulate the phenomena are correct also. Validation is confirmed by comparing the results with experimentally obtained data [13].

The difference in testing between scientific and commercial applications is only in the tests design because scientific application tests are more focused on numerical precision and proper use of mathematical libraries and their implementations, while the concepts of regression and automated testing remain the same [3]. Scientific applications testing means mainly testing the simulation models. The hardest task in this process is the definition of test cases, since the results that have to be obtained are usually not known from start. A good practice is to make a collection of test problems in the scientific community, i.e. to create benchmark tests that have already analyzed solutions which can be used as expected results [18] [3]. Good recommendations for generating test cases are to generate test cases based on code specification documents, to take into account the analysis of the limit values, to create test cases based on assumptions about errors and certainly at least 90 % of code statements to be included [4]. Due to frequent requirements and source code changes, regression testing that is performed after each made change can detect possible errors. The tests will help also to detect whether the change in the code caused another change that will affect the accuracy of the calculations ant it will help to find if the mistakes are in the range of tolerance [10]. The following testing methods can be also applied in the process of testing: module testing (testing smaller isolated sections of code), system testing (testing combination of modules), mutation testing (testing mutated code) and static testing (testing the code by checking each line manually) [18]. An additional challenge during the testing of scientific applications is to separate the errors in terms of whether they belong to the software or to the model and mathematical approximation [6].

Checking the accuracy of the results obtained from scientific applications cannot be done in the same way as in commercial software applications, so testing is conducted in a different way, primarily because most of the requirements are non-functional [3]. There is an evidence that shows that even 40% of the reasons for the decline of the software can be eliminated with the static analysis of errors [14]. The process of detecting errors can be enhanced by using predictive models that based on mathematical models and number and types of errors that occurred in previous versions could predict the errors that may occur in the next software version. There are some tools that automate process of writing tests, some are used for displaying the results, automation and tests management [4].

4 The Survey

4.1 HP-SEE

The HP-SEE project (High-Performance Computing Infrastructure for South East Europe's Research Communities) provides a common electronic infrastructure that connects HPC centres in South East Europe and enables new countries from this region to be included in the project also. This project creates opportunities for scientists and research communities in South East Europe to include their projects in any of the three available research communities: computational physics, computational chemistry and live sciences. In addition, it increases the process of collaboration between scientists and thus contributes to better research quality and achievements.

The success of the project is confirmed by 26 running applications in the three scientific communities and involvement of 14 countries from Southeast Europe region. There are 12 applications from 7 countries in the computational physics community, 7 applications from 6 countries in the computational chemistry community and 7 applications from 5 countries in life sciences community. Furthermore, it is motivation for other research centers and scientists to engage their projects and become part of this research infrastructure [17].

4.2 Results

In our quest to become more relevant, we conduct a survey where 12 scientists from the closed circle of people involved in the HP-SEE project gave answers to 29 questions, especially in the area of testing. These questions are written in accordance with the offered literature and research about software testing. It contains multiple choice and one choice questions about testing activities, testing models, ways of conducting the tests, testing tools, formal testing approaches, etc. The question short descriptions and answers from the survey are shown below.

Q1: Importance of testing process: 7 answered *Very important*, 4 *Extremely important*, 1 *Neutral*, 0 *Not important*, 0 *Not at all important.*

Q2: People responsible for testing process: 11 answered *Original developer*, 4 *Special software testing team.* This means that 3 of them chose the both options.

Q3: Defects in existing software: 6 answered *Not sure*, 4 *Yes*, 2 *No.*

Q4: Features that have to be tested specified in advance: 10 answered *Yes*, 2 *No.*

Q5: Features that have not to be tested specified in advance: 10 answered *No*, 2 *Yes.*

Q6: Use of automated testing: 12 answered *No*, 0 *Yes.*

Q7: Types of used testing methods: 11 answered *White box testing methods*, 7 *Black box testing methods.* It means that 6 of them chose the both options.

Q8: Types of used test activities: 7 answered *Test-last Development*, 3 *Not specified*, 2 *Test-first Development.*

Q9: Description of test cases: 6 answered *Freely*, 3 *In a formal text-based language*, 3 *No test cases described*, 0 *With help of standardized forms.*

Q10: Test cases generation technique: 6 answered *No explicit test case generation technique was used*, 5 *Model-based techniques*, 5 *Source code analysis*, 4 *Boundary value analysis*, 1 *Category partitioning.*

Q11: Used testing types: 8 answered *Integration testing*, 8 *Functional testing*, 7 *System testing*, 6 *adHoc testing*, 5 *Unit testing*, 2 *Regression testing*, 1 *Recovery testing*, 0 *None of the Above.*

Q12: Used static analysis techniques with tool support: 6 answered *Integration testing*, 3 *Control flow analyses*, 3 *Data flow analyses*, 3 *Static analyses of models*, 2 *Checking of coding standards*, 1 *Static analysis of texts/documents*, 1 *Other tool-supported static analyses.*

Q13: Used test design techniques based on experience: 5 answered *Other test techniques based on experience*, 4 *Exploratory testing*, 3 *No test techniques based on experience*, 2 *Error guessing*, 1 *Fault attack with defect checklists.*

Q14: Used test design techniques based on structure: 8 answered *No test techniques based on structures*, 2 *Statement coverage*, 2 *Branch coverage*, 2 *Condition coverage*, 1 *Path coverage*, 1 *Other test techniques based on structure*, 0 *Multiple condition coverage.*

Q15: Use of testing tools to perform testing: 8 answered *No*, 4 *Yes.*

Q16: Type of used testing tool in the testing process (if any): 4 answered *Source code testing tool*, 3 *Functional testing tool.*

Q17: Importance of requirement coverage: 4 answered *Very important*, 4 *Neutral*, 3 *Not important*, 1 *Not at all important*, 0 *Extremely important.*

Q18: Tracking the requirements coverage: 8 answered *No*, 4 *Yes.*

Q19: Importance of release version tracking: 4 answered *Very important*, 4 *Not important*, 3 *Neutral*, 1 *Not at all important*, 0 *Extremely important.*

Q20: Release version tracking implementation: 9 answered *No*, 3 *Yes.*

Q21: Importance of testing documents generation: 6 answered *Very important*, 4 *Neutral*, 2 *Not important*, 0 *Not at all important*, 0 *Extremely important.*

Q22: Testing documents generation: 7 answered *No*, 5 *Yes.*

Q23: Created test deliverables: 6 answered *Test plan*, 5 *Test cases*, 5 *Test reports*, 4 *Test scripts*, 3 *No test deliverables*, 0 *Defect/enhancement logs .*

Q24: Linking requirements to the tests that verify them: 6 answered *Yes*, 6 *No.*

Q25: Benefit of creating graphs and reports which outlines the state of system testing: 10 answered *Yes*, 2 *No.*

Q26: Validation of code by comparing simulation outputs with the results of physical experiments: 10 answered *Yes*, 2 *No.*

Q27: Barriers for testing: 8 answered *Time*, 3 *Costs*, 4 *No barriers for testing*, 1 *Available corresponding experiment data*, 0 *No support from the high-level management.*

Q28: Criteria to decide that the testing is completed: 10 answered *All test cases are executed without finding more bugs*, 8 *No bugs are found any more*, 2 *When code coverage analysis are reached*, 1 *Informal*, 1 *Coverage of tested models*, 1 *When results are validated using analytical or other solutions.*

Q29: Found defects: 11 answered *Corrected in accordance with its priority*, 1 *Allocated to a defect class*, 0 *Allocated to a priority*, 0 *Other.*

5 Software Engineering Practices for Creating Higher Quality Scientific Applications

The survey results helped us to determine in detail the shortcomings of software engineering formal practices and methods in the development of scientific applications and to propose specific quality improvement suggestions. Responses from the survey were also used to create a correlation between the questions and to discover potential dependencies that will contribute to improving the perceptions and practices in the development process of scientific applications. Correlations among questions refer primarily to related testing activities applied by development teams, but more specifically to the techniques and models that are used to generate test cases, error detection, types of testing, analysis, etc.

Some of the correlations are: Teams that perform unit testing also perform integration testing; Teams that perform release version tracking perform regression testing; Teams that track the requirements coverage perform functional testing; Teams that do not have testing plan and documentation perform adHoc testing; Teams that do not have any testing documents describe the test cases freely and do not link the requirements to tests that verify them; Teams where only original developer is responsible for testing do not use structure-based test design techniques and testing tools often.

The results showed that the process of testing is very important for scientists but the fact that 50% of them are not sure if their software has bugs indicates that the process of testing has major drawbacks. Given that most of the scientists answered that only software developers are responsible for the testing process, the need for inclusion of software engineers in the process of testing is imminent.

To improve the quality of scientific applications we propose a model that covers the following three aspects in the testing process: using well-defined practices, using formal methods and generating documents, certainly with the possibility of minor changes in order to adapt to the specific requirements of the application.

In terms of the practices for defining requirements first thing that needs to be created is a requirements formal specification. Although some of the requirements may change or occur in any of the later stages of development, all of them should have appropriate description using templates proposed by software engineering. That will provide a better overview of the features to be tested later. Irregularity in the process of requirements defining is confirmed by the survey where scientists are mostly neutral in terms of coverage requirements in the testing process and most of them do not practice that.

Well-specified requirements can significantly facilitate the process of defining test cases by providing greater visibility and coverage of the features that have to be tested. The survey results confirm that the features that have to be tested are written in advance in most applications, but the test cases are not created by linking them to the specified requirements, but test cases are mostly written in free form. Even some of them are not documented and the process of their generation does not use any specific technique. These principles must be changed with some software engineering practices like using formal language to define test cases, generating test cases by using specific techniques based on models, based on analysis of the boundary values or analyzes of the source code, etc.

The process of testing automation and use of tools that allow source code or functionality testing are very important and can greatly improve and accelerate the process of testing. To improve the quality of applications, the current practice of manual testing must be changed. Generally, today there are many modern tools, that support and perform various actions such as control and data flows analysis, static analysis of code and documents, checking coding standards etc.

Testing process must include various methods in order to provide complete assurance that the software is error free. According to the survey, most of the tests are performed on the source code and mostly after the software is developed. Thus, the integration testing is applied to check the functioning of the system after the integration of modules and functional testing which is based on requirements. However, to implement integrations testing, first the testing of each module should be performed (unit testing) and to implement functional testing all requirements should be specified and linked to test cases. Applying the regression testing or testing performed after each change is a good approach due to the frequent changes that occur. Agile software testing could be very useful testing method, especially when incremental and iterative development approaches are used. Testing the entire system and testing after system crashes or hardware error are very important also.

Generating documents across the entire development process of scientific applications will contribute to raising the quality of applications because it will allow better planning and scheduling activities, greater visibility and insight about what has been done and what should be done in the future. The results showed that there are no generated test documents in the development of some scientific applications, which means that the risk of errors is much higher because no detailed inspection of the features and no reports exist. The lack of practice to create documents must be replaced by using templates for test documents generations such as test plans, test cases, test reports, error logs, etc.

6 Conclusion

In this paper we have presented the software engineering principles and practices that can improve the scientific applications quality. The research has shown that many formal and standardized software practices and principles can be adapted to scientific software development process. The paper especially outlines the practices and principles that can be used in the testing process in order to ensure that errors have been removed and that the complex numerical calculations have been performed with great accuracy and precision. The survey we conducted helped us to understand the basic deficiencies in the scientific applications development process and that give us guidelines for better suggestions of software engineering practices that would improve not only the process of testing, but the overall quality of the applications. The proposed practices can be adjusted to the development process of any scientific application. A further expansion of these recommendations can be creation of a unified model that will be able to perform a detailed examination of the characteristics of a scientific application and therefore will be able to classify the problem in a given category and propose more specific practices.

References

1. Andersen, P.B., Prange, F., Serritzlew, S.: Software engineering as a part of scientific practice (April 2011), `http://imv.au.dk/~pba/Homepagematerial/publicationfolder/softwareengineering.pdf`
2. Basili, V.R., Carver, J., Cruzes, D., Hochstein, L., Hollingsworth, J., Shull, F., Zelkowitz, M.V.: Understanding The High Performance Computing Community: A Software Engineer's Perspective. IEEE Software 25, 29–36 (2008)
3. Baxter, R.: Software Engineering Is Software Engineering. In: Proceedings of the First International Workshop on Software Engineering for High Performance Computing System Applications, pp. 14–18. IEEE Computer Society, Washington, DC (2004)
4. Chin, L.S., Worth, D.J., Greenough, C.: A Survey of Software Testing Tools for Computational Science. Technical Report RAL-TR-2007-010, Software Engineering Group, Computational Science & Engineering Department, Rutherford Appleton Laboratory, Oxfordshire, UK (2007)

5. Eijkhout, V., Chow, E., Geijn, R.: Introduction to High-Performance Scientific Computing. Public Draft (July 2010), http://www.math-cs.gordon.edu/courses/cps371/doc/scicompbook.pdf
6. Hannay, J.E., MacLeod, C., Singer, J., Langtangen, H.P., Pfahl, D., Wilson, G.: How Do Scientists Develop and Use Scientific Software. In: Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering, pp. 1–8. IEEE Computer Society, Washington, DC (2009)
7. Howison, J., Herbsleb, J.D.: Scientific software production: incentives and collaboration. In: Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work, pp. 513–522. ACM, New York (2011)
8. Jurgens, D.: Survey on Software Engineering for Scientific Applications- Reusable Software, Grid Computing and Application. Institute of Scientific Computing. Technische Universitat Braunschweig, Germany (2009), http://rzbl04.biblio.etc.tu-bs.de:8080/docportal/servlets/MCRFileNodeServlet/DocPortal_derivate_00006306/Juergens-Survey-Software-Eng-Scientific-Applications.pdf
9. Loh, E., Van De Vanter, M.L., Votta, L.G.: Can Software Engineering Solve the HPCS Problem? In: Proceedings of the Second International Workshop on Software Engineering for High Performance Computing System Applications, pp. 27–31. ACM, New York (2005)
10. Neely, R.: Practical Software Quality Engineering on a Large MultiDisciplinary HPC Development Team. In: Proceedings of the First International Workshop on Software Engineering for High Performance Computing System Applications, pp. 19–23. IEE, Stevenage (2004)
11. Phadke, A.A., Allen, E.B.: Predicting Risky Modules in Open-Source Software for High-Performance Computing. In: Proceedings of the Second International Workshop on Software Engineering for High Performance Computing System Applications, pp. 60–64. ACM, New York (2005)
12. Post, D.E.: The Challenge for Computational Science. In: Proceedings of the First International Workshop on Software Engineering For High Performance Computing System Applications, pp. 8–13. IEEE Computer Society, Washington, DC (2004)
13. Post, D.E., Kendall, R.P.: 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 18, 399–416 (2004)
14. Roy, C.J.: Practical Software Engineering Strategies for Scientific Computing. In: Proceedings of the 19th AIAA Computational Fluid Dynamics Conference, pp. 1473–1485. American Institute of Aeronautics and Astronautics, Inc., Reston (2009)
15. Segal, J.: Models of scientific software development. In: First International Workshop on Software Engineering in Computational Science and Engineering, SECSE 2008, Leipzig, Germany (2008)
16. Segal, J.: Scientists and Software Engineers: A Tale of Two Cultures. In: Proceedings of the Psychology of Programming Interest Group, pp. 44–51. University of Lancaster, UK (2008)
17. The official HP-SEE site, http://www.hp-see.eu/
18. Zheng, B.: Documentation Driven Testing of Scientific Computing Software. Master Thesis, McMaster University, Ontario, Canada (2009), http://digitalcommons.mcmaster.ca/cgi/viewcontent.cgi?article=5421&context=opendissertations