Framework for Developing Scientific Applications: Solving 1D and 2D Schrödinger Equation by using Discrete Variable Representation Method

Conference Paper · April 2015

DOI: 10.13140/RG.2.1.1393.5529

3 authors:

Bojana Koteska
Ss. Cyril and Methodius University in Skopje
64 PUBLICATIONS **326** CITATIONS
SEE PROFILE

Anastas Mishev
Ss. Cyril and Methodius University in Skopje
96 PUBLICATIONS **517** CITATIONS
SEE PROFILE

Ljupco Pejov
Ss. Cyril and Methodius University in Skopje
158 PUBLICATIONS **1,596** CITATIONS
SEE PROFILE

Framework for Developing Scientific Applications: Solving 1D and 2D Schrödinger Equation by using Discrete Variable Representation Method

Bojana Koteska and Anastas Mishev

Faculty of Computer Science and Engineering,
1000 Skopje, Republic of Macedonia
e-mails: {bojana.koteska,anastas.mishev}@finki.ukim.mk

Ljupco Pejov

Faculty of Natural Science and Mathematics,
1000 Skopje, Republic of Macedonia
e-mail: ljupcop@iunona.pmf.ukim.edu.mk

***Abstract*—The absence of software engineering practices while developing scientific applications has negative impact on the quality of the applications. As a result, the probability for finding bugs in the application is higher, testing is more difficult and further code optimization and paralelization become an issue. In order to improve the developing process, in this paper, we propose a framework for developing scientific applications. The framework helps scientists to understand some of the basic concepts of software engineering and to change their current habits for developing scientific applications. Our goal is to adapt and modify some of the software engineering practices in every phase of the application development process. Aiming to use this framework in practice, we apply the recommendations for all phases while developing application for solving 1D and 2D Schrödinger equation by using the Discrete Variable Representation method (DVR). Using the framework resulted in better code organization, linked execution of the application modules for 1D and 2D equations, defining requirements and designing tests. As a final product we have an application organized in modules, documentation for each developing phase, comments in the code and executable tests.**

Keywords–Scientific application; Software Engineering; Framework; Schrödinger equation; Software quality.

I. INTRODUCTION

A scientific application is a software application that simulates activities from the real world and turns objects into mathematical models [1]. Scientific applications are designed to perform numerical simulations of natural phenomena in different scientific fields: computational chemistry and physics, informatics, mathematics, bioinformatics, etc. The execution of such applications that perform simulation of scientific experiments with large amount of data requires powerful supercomputers, high performance computing and Grid computing [2].

According to the Institute of Electrical and Electronics Engineers standard (IEEE Std) 610.12-1990, software engineering is defined as an application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, that is, the application of engineering to software [3]. The main problem related to the development of scientific applications is the current development practice. In our previous paper [4], we conducted a survey among scientists - participants in the High Performance - South East Europe (HP-SEE) project and we found out that they do not use software engineering practices in the scientific application development process.

Scientific applications development differs from the development of commercial applications, especially in the stage of testing. Scientific applications are developed by the scientists themselves and used in the scientific research group which means that software can not be tested based on users' requirements, but the results can be compared to the results obtained from the real experiments or they are based on theory. High performance computing applications, such as scientific applications, can basically be the same with any kind of application, but only small number of additional changes in terms of planning, requirements elicitation, testing and development approaches are required [5]. It means that the same development practices cannot be fully applied, but there is a possibility to modify the practices and to make some adaptations.

The goal of this paper is to propose a development framework that will help scientists to change the current development practices and to show that software engineering can be included in the development process of scientific applications. The development framework provides basis for a complete scientific software development process and it also gives some quality recommendations that can be applied in the development process. The framework is generic which means that steps for developing can be used for different scientific applications. It provides a set of rules, recommendations and software engineering development practices. The main contribution of this paper is the description of the full development process of the scientific applications which tries to adapt and modify the existing software engineering practices for developing commercial applications.

The possibility of inclusion of the development practices proposed in the framework is validated with the development of an application for solving 1D and 2D Schrödinger equations by using the DVR numerical method. The application is developed by following the suggested development stages and recommendations which means that requirements are written, architecture is designed, code is organized in modules and optimized, testing is automated, etc.

The paper is organized as follows: related work is presented in the Section 2. Section 3 describes the current development practices used in scientific application development process. The framework is specified in the Section 4. The development process that includes the development stages proposed in the framework which are used for programming the application for solving 1D and 2D Schrödinger equations by using the DVR

 ISBN: 978-1-61208-449-7

numerical method is given in Section 5. The conclusion and future work are provided in the Section 6.

II. Related Work

There are several papers that emphasize the need of software engineering when developing scientific applications, but no paper describes the full development process. In [6], the author presents the most modern software engineering practices relevant to scientific computing. He gives an overview of some software development models, choice of programming language, static analysis tools, dynamic testing procedures, version control systems, software quality and reliability. Here, there are some useful software engineering practices in the scientific applications development [7]: identification of resources required to develop the application; developing plan and schedule for completion of tasks and responsibilities; managing requirements, documenting them and constant control; making test plans and documentation; managing changes; managing the risks that may arise during development; constant quality control.

The quality of scientific applications can be improved by using generic programming, domain modeling and component based programming. For example, the creation of a meta model with established rules assists in the process of defining the application model needed to solve problems specific to that domain [8]. Software engineers can help scientists to realize the benefits of reusing by providing to them software frameworks. The framework will reduce the effort and time required for development, especially if scientists already have some knowledge about the used technologies, e.g., Message Passing Interface (MPI) [9]. Open Community Engagement Process is a model for software development which brings the software engineers and scientists together. The model define four steps: design, develop, refine, publish. This process is iterative and it follows incremental development approach and agile principles [10]. There is also a project for creating software infrastructure for scientific computing (ACTS) that provides a lot of free software tools which are divided into four categories: numerical calculations, code development, code execution and development libraries. These tools provide support for solving linear systems, optimization, obtaining analytical solutions, visualization, etc. [11]. Some practices that can improve the scientists' productivity and reliability of scientific applications are recommended in [12]. The survey we presented in [4], helped us to find some shortcomings in the development process of the scientific applications. Based on it, we proposed some practices for increasing the quality of the applications.

III. Scientists' development practices

Scientists usually learn programming independently or they are taught by other scientists. They believe that the process of software development is only the process of coding. The applications they are developing are intended for their scientific group or closer scientific community. The most important thing for scientists is to get scientifically correct results [9][13]. If the output results are correct, they are not very interested in making additional optimizations or parallizations. They are often guided by the thought that if a hypothesis is proven once, there is no need for reprogramming that section [5].

The scientists write codes in small teams without previous formal training [6]. The reason for the poor quality of high-performance computing applications lies in the lack of used software engineering formal methods and practices [9][14] . It may result in a larger number of errors, difficult understanding of the code written by the other scientists in the community, using additional unnecessary resources, problems with refactoring and optimizations later, etc. [13].

Scientists do not practice writing requirements or any kind of documents and they believe that requirements should only be discussed, but not written [13]. Later, when the testing is performed, there is no information what is done and what have to be done more. When the process of verification starts, scientists primarily give importance to the quality of the algorithm rather than its realization [9]. Usually the scientific problems have no precise solution and numerical methods are used to perform an approximation. Often, the verification means comparison with experimental results [6]. The software verification is based on professional judgment, such as comparison of the model output with other model outputs (benchmark) which will not always provide a consistent comparison and visual comparison between two pictures [15]. A primarily goal for scientists is to test the implemented theory, not the algorithm implementation. The testing is not the most important thing until an error that affect the correctness of scientific results appears [14].

The scientists in the HP-SEE project that participated in the survey [4] said that testing is a very important process, but mostly only the original developer is responsible for testing. Also, they are aware of their application errors, but they test manually, which means that they do not use any testing tools. Scientists think that writing testing documentation is very important process, but they describe the test cases freely. They answered that the biggest barrier for testing is time. Scientists are interested in graphs and/or reports that outline the state of system testing.

Taking into account all the problems mentioned above (difficult testing, no documentation, poor code comments, problems with optimization, etc.), we decided to propose a framework that will guide the scientists through the development process of the scientific applications.

IV. Framework for Developing Scientific Applications

This section describes the stages for developing scientific applications. We propose the incremental software development model which is elaborated in Software Engineering: 9th edition by Ian Sommerville [16]. We made small modifications of the model because there are two main differences between standard software development and scientific applications: The results from the scientific applications are verified by comparison to the results obtained from physical experiments; There are no customers that can evaluate the application since applications are developed only for the scientific community. We adapt this model by making changes in the process of evaluation, in test-driven design and short reports after each iteration.

This development process is characterized by fast delivery of increments and their evaluation which helps the cost of changing requirements to be reduced. Each increment of the development process contains 9 phases: **planning, requirements definition, system design, test cases design, coding,**

 ISBN: 978-1-61208-449-7

testing, evaluation, writing short report. It is shown in Figure 1.

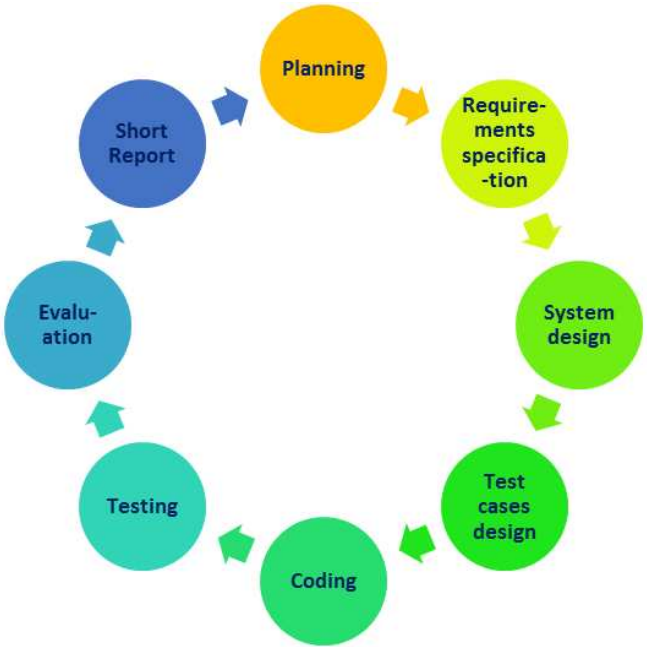

Figure 1. Scientific application development process

A. Planning

Each increment begins with the planning phase. The plan is a non-formal document which contains a list of activities that should be performed in the next iteration. For example, scientists should think about what parts (modules) have to be coded in that iteration, how they will be tested and evaluated. If there are more scientists who develop the application, the tasks should be assigned appropriately.

B. Requirements Specification

This is the development phase where requirements should be written formally. The document containing formal requirement specification is needed for better evidence of completed and future activities. We divide the requirements in two categories: functional requirements and nonfunctional requirements.

A **functional requirement** is a requirement that describes an application functionality (also inputs and outputs of a function). A **nonfunctional requirement** is a requirement that specifies the system behavior (constraints).

Each requirement should be consisted of the following fields: Id - unique requirement identifier; Name - short name that describes the requirement; Requirement type - functional or nonfunctional requirement; Version - current version of the requirement (as a number); Description - description of the functionality that needs to be realized (if functional) or description of tests and evaluation of the application functionalities or any hardware and software requirement; History of changes in each version of the requirement which is needed because of frequent requirements' changes, especially when solution goes in the wrong direction. The requirement specification will provide a better overview of the features that need to be tested, and less problems, such as delayed tasks realization, no working plan, late and incomplete testing, errors, etc.

C. System design

System design is the part where the hardware and software systems requirements are specified in order to establish a system architecture (for example, if the application needs multicore processor, Grid, high-performance computing to be run or some libraries that provide parallel execution). In this section, compilers, integrated development environments (IDEs), platforms and operating systems should be also specified.

D. Test cases design

Test cases designs are mostly related to functional requirements which means that well specified requirements can contribute to better design of test cases. Test cases need to be defined by using standardized forms. Also, boundary values and source code analysis could be considered as a relevant information. Test cases where the correct value cannot be specified should contain a range of values.

Each test case description should include the following fields: Id - a unique test case identifier; Name - name of the test case; Requirement id - specific id of the requirement tested with this test case; Goal - a description of the goals of the test case; Preconditions - conditions that must be met in order to perform the test case (results from other tests or some additional conditions); Execution environment of the test case; Expected results; Actual results; Test case status (pass or fail); History of the changes in each test case version. Code and branches coverage techniques could be useful for generating test cases.

E. Coding

The best approach is to define more independent modules because they can be used as generic functions in many other applications from the same or similar scientific domain. If any part of the code is repeated it definitely should be written as a separate function. The declaration and definition of unused variables should be avoided. Also, release of the memory should be performed always when possible. Comments must be written through the code and, if possible, some optimizations can be done. Parallelization can be performed by using some libraries (for example, OpenMPI [17], if the code is written in C).

F. Testing

When test cases are created and code is written it is time to make and run tests. There are various methods that can be used to provide assurance that the software is error free. White-box testing includes tests designed to check the source code. If possible, tests should be created before the software is developed. Only non redundant test cases should be selected, the other should be eliminated. The following criteria are good indicators that testing process is completed: all tests are performed successfully without errors, the criteria for source code coverage and test models are satisfied and validation by analytic solutions is achieved.

Tests automation and tools that provide source code testing or functional testing are very important and can greatly improve and speed up the process of testing. To improve the

 ISBN: 978-1-61208-449-7

quality of applications, the current practice of manual testing should be changed. There are many frameworks for creating and running tests (for example, frameworks for C code: Check, CUnit, AceUnit, CuTest, etc.).

We recommend white box testing for each module, and then integration testing to check the interconnection between the modules and functionality of a system as a complete product. The possible conflicts with the already created tests must be resolved by changing the tests.

G. Evaluation

The results can be evaluated only by a limited number of users (usually scientists themselves) because the accuracy of the results is often based on theory and experiments. Scientists who have developed applications that solve similar problems can help to find errors. Developing experience and learning techniques for guessing or past errors can help a lot in the test design. Found errors should be sorted by priority, for example, if the error affects the further development or it is a critical error, it should be marked as error with higher priority and corrected as soon as possible.

H. Writing Short Report

A short report is document that describes the finished tasks in the current iteration. The tasks done in each phase should be listed. This is very important when a new member joins the team because he/she will know what is done and what have to be done. Scientists do not have practice for generating documents, but creation of documents can help to further improving and upgrading the application.

V. Solving 1D and 2D Schorödinger Equations by using Discrete Variable Representation Method

Discrete Variable Representation methods are widely used in different scientific domains, such as chemical physics, molecular quantum dynamics, etc. DVR's are described as a representation whose basis functions are localized about discrete values of the variables. DVR's are also approximations of coordinate operators by their values at the DVR points which are assumed diagonal. DVR methods are acceptable for many problems because they simplify the calculation of the kinetic energy matrix elements of the Hamiltonian matrix and also potential matrix elements which are the value of the potential of the DVR. DVR's provide efficient numerical solutions to quantum dynamical problems. When the DVR's product in multi-dimensional systems is calculated, operation of the Hamiltonian on a vector is fast and the Hamiltonian matrix is sparse (with many 0's) [18].

Schrödinger equation is a partial differential equation that describes the dynamics of system at atomic and molecular level. A time independent equation is represented with the following formula:

$$H * \Psi = E * \Psi \quad (1)$$

where H is the Hamiltionian operator, Ψ is the wave function of the quantum system and E is the energy of the state Ψ. A time dependent Schrödinger equation has the form:

$$i * \hbar \frac{\partial \Psi}{\partial t} = H * \Psi \quad (2)$$

where H is the Hamiltionian operator, Ψ is the wave function of the quantum system, $\hbar$ is the Planck Constant divided by 2Π and $\frac{\partial}{\partial t}$ is a partial derivative with respect to time t.

We have to develop scientific application for solving 1D and 2D Schrödinger equation. We will organize the development process in increments. There are already some codes for solving 1D and 2D Schrödinger equation in Mathematica, but we want to use C language because our goal is to use this module as an independent part in more complex application written in C. Detailed descriptions of the requirements and modules are given in the following subsection.

A. Increment 1

1) Planning:

- define the inputs and outputs of the module for solving 1D and 2D Schrödinger equation
- define developing environment, software and hardware needed for developing and execution of the application
- split the algorithm in modules
- define inputs and outputs for each module
- create tests for modules by choosing any tool for test automation
- develop all submodules needed for the module for solving 1D and 2D Schrödinger equation
- perform tests and evaluate the results
- correct errors if any

2) Requirement Specification: In this subsection, we will provide only the list the requirements with their IDs, but they also have to be specified as described above.

Functional requirements:

1) Module for multiplication of two 2D arrays (input - two 2D arrays of type double and number of rows/columns of the array (matrices are quadratic) of type integer, output - one 2D array of type double).
2) Module for making a diagonal 2D array(input - 1D array of type double and number of rows/columns of the array of type integer, output - diagonal 2D array of type double).
3) Module for multiplication of a scalar and a 2D array (input - 2D array, scalar and number of rows/columns of the array, output - 2D array).
4) Module for making an identity 2D array (input - the number of rows/columns of the array, output 2D array).
5) Module for addition of two 2D arrays (input-two 2D arrays and number of rows/columns of the array (matrices are quadratic), output - one 2D array).
6) Module for making a transposed 2D array (input - the number of rows/columns of the array, output 2D array).
7) Structure for a file row which contains three double numbers.
8) Structure for a file which contains array of elements of type "structure for a file row" and number of rows of the file (integer).
9) Module for reading data from file (input - char array(file path), output- element of type "structure for a file".

 ISBN: 978-1-61208-449-7

10) Module for sorting rows in file (input - element of the type "structure for a file", output - element of the type "structure for a file").
11) Module for calculating eigenvalues (input - 2D array of type double and number of rows/columns of the array(integer), output- 1D array of type double).
12) Module **thcheby** for calculating array of x-values, y-values(2D), transformation matrices for x and y values in finite basis representation (FBR) (input - number of points for x values- integer, number of points for y values - integer, minumim and maximum values for x and y-double, output - 1D array for x points of type double, 1D array for y points of type double, 1D array of x points in FBR of type double, 1D array of y points in FBR of type double, 2D array for transforming x points from FBR to DVR of type double, 2D array for transforming y points from FBR to DVR of type double. y-values are only needed for solving 2D Schrödinger equation.

Nonfunctional requirements:

1) Algorithm for array sorting should have complexity smaller then $O(n^2)$.
2) Application should be scalable (for example, when input data size increases, dynamic memory allocation must be used).
3) Memory used by the objects should be released when they are not used anymore.

3) System Design: The application for solving 1D and 2D Schrödinger equation can be run on a single processor machine. A C compiler is needed for a code compailing. A code editor should be installed also. The application can be run on any operating system. The results are provided to the standard output. In order to perform the testing, the CuTest framework for testing C codes should be installed also [19].

If the user wants to solve 1D or 2D Schrödinger equation, he/she has to provide only the input parameters to the module for solving these equations and file path.

4) Test Cases Design: In order to test the system for each specified functional requirement, we defined test case, as specified in the Section 4. Writing test cases before writing the code will help us to identify the needed input, predicted results and conditions. This is the part where only text document is written and later in the testing phase we will use the CuTest framework to write and run the unit tests. For example, the test case for the 12-th functional requirement (**thcheby** module) specified above, has the following conditions:

1) **deltax** is the difference between the maximum(**xmax**) and minumum value of x(**xmin**);
2) **deltay** is the difference between the maximum(**ymax**) and minumum value of y(**ymin**);
3) **nx1** is the number of x points increased by 1;
4) **nxy** is the number of y points increased by 1;
5) The i-th member of the array of x values (**ptsx**) has the value $((i+1)*deltax*1.0)/nx1+xmin$
6) The i-th member of the array of y values (**ptsy**) has the value $((i+1)*deltay*1.0)/ny1+ymin$
7) The i-th member of the array of x values in FBR (**fbrtx**) has the value $(((i+1)*\Pi)/deltax)^2$;
8) The i-th member of the array of y values in FBR (**fbrty**) has the value $(((i+1)*\Pi)/deltay)^2$;
9) The element at the position (i,j) of the transformation matrix for x values (**Tx**) has the following value $\sqrt{2.0/nx1}*\sin((i+1)*(j+1)*\Pi/nx1)$
10) The element at the position (i,j) of the transformation matrix for y values (**Ty**) has the following value $\sqrt{2.0/ny1}*\sin((i+1)*(j+1)*\Pi/ny1)$

For each condition a status should be written also (PASSED/FAILED).

5) Coding: The code is organized in modules. The coding of a module is performed after the specification of the test case for that module. Arrays are defined by using pointers and memory is released always when possible. The calculation of eigenvalues is performed by using the GNU Scientific Library(GSL) library. The complexity of the algorithm is $O(n^2)$. Sorting was implemented by using the quick sort method.

One of the most important things that scientists do not practice is writing comments through the code. We add comments for describing the modules, variables, cycles and statements. Another useful thing in programming is the concise naming of the variables and methods.

6) Testing: Testing was performed by using the CuTest system which is designed for writing, administering, and running unit tests in C [19]. A test case is passed if all conditions for that test case are satisfied. In order to check the correctness of the code, assertions must be added. For example, in order to check the correctness of the test case for the 14-th functional requirement, we have to write several test functions and to call them in the main function. In Figure 2, function for testing the members of the array of x values in FBR (**fbrtx**) is shown, where i-th member has the value $(((i+1)*\Pi)/deltax)^2$;, as described in the subsection test cases above.

```
void TestFbrtxMembers(CuTest *tc)
{
struct return_objects result=
thcheby(10, 1, 5, 10, 2, 6);
double *ac=result.fbrtx; //actual result
int i;
double *ex=
malloc(10*sizeof(double));//expected result
for(i=0;i<10;i++)
{
ex[i]=square(((i+1)*M_PI)/(5-1));
CuAssertTrue(tc,abs(ex[i]-ac[i])<0.00001);
}
```

Figure 2. Descriptive Caption Text

7) Evaluation: Evaluation was performed by comparing the results from our application to results from the provided code in Mathematica by the Upssala University. When comparing experimental results, another way to test this application to test the convergence of the results with increasing the density of grid points which is equivalent to increasing the number of basis functions. Several errors in the modules were found and they were corrected.

8) Writing Short Report: In the first increment, all modules specified in the requirements section needed for calculating

 ISBN: 978-1-61208-449-7

were written and tested. The errors were corrected and all tests passed successfully. Also, all nonfunctional requirements were taken into consideration.

B. Increment 2

1) Planning:

- adapt the input for 1D and 2D equation
- integrate all modules into one module for solving 1D and 2D Schrödinger equation
- create tests for the module for solving 1D and 2D Schrödinger equation
- perform tests and evaluate the results
- correct errors, if any

2) Requirement Specification: **Functional Requirements**

- Module for making the interpolation function (**PES**) and energy list (**pel**)
- Module for calculating 1D and 2D Schrödinger equation: Input - Potential energy values computed on 2D grid of points (E, x, y) or 1D grid (E,x) read from a separate file, the number of DVR points, minimum and maximum values of x(1D) and x, y(2D), interpolation function and mass; Output - Frequencies of various vibrational transitions $(n, m) - > (l, k)$ to be compared with experiment, but also the convergence with respect to computation-related parameters to be tested, - Vibrational wavefunctions on 2D grid of points (ψ, x, y) or the square-modulus of ψ on 2D grid of points $(|\psi|^2, x, y)$, i.e. $((\psi * \psi), x, y)$.

3) System Design: Same as specified in Increment 1.

4) Test Cases Design: Two test cases were created. The conditions included in the test case for the first module in this increment are for interpolation function which should be implemented by using the Hermite interpolation method and some details about making the pel array. The second test case only checks the output results because all modules that are called in this module are checked in the previous increment. The only important thing here is the proper modules integration.

5) Coding: In this increment, only the developed modules were called in proper order in the module for calculating 1D and 2D Schrödinger equation. The module for calculating the 2D Schrödinger equation was developed. Also, this module is appropriate for solving 1D Schrödinger equation by eliminating the calculations which include the second coordinate y. The most convincing part here was programming of the interpolation method which is an integrated function in Mathematica.

6) Testing: Testing was also made by using the CuTest framework. Two tests were created and run.

7) Evaluation: The results were compared to the results from the program written in Mathematica.

8) Writing Short Report: Time needed for this increment was shorter because all modules were tested and programmed in the previous increment. An algorithm for Hermite interpolation was programmed and some errors that were found in the integration function were corrected.

VI. Conclusion and Future Work

In this paper, we proposed a framework for developing scientific applications. The framework describes the phases of the development process. In order to prove the developing model, an application for calculating 1D and 2D Schrödinger equation was developed. The results show that the framework is suitable for developing this application because at the end we have understandable code organized in modules, documentation and generated tests which are shortly described in the Section 5. The existing solutions for this problem do not have comments through the code or any documentation and it is very hard to understand the code. There are no proposed developing stages for scientific applications which will guide scientists to write documents and to plan the development process. This framework will help scientists to change the current development practices and will provide better overview of the application for the scientists who will join the project later. The framework is a guide for developing scientific applications because it explains the developing steps in details. The independent modules or complete program as a module can be used also in other scientific applications. Although there are many scientific libraries, usually a scientific research group needs modules with specific input, output and parallel programming methodologies which are originally developed by the group. Our goal is to test the framework of the large scientific applications and to check if this development process can be applied for different scientific applications. Also, we want to provide a set of documented modules written in C that are used in many different scientific applications which will be available for the scientists to use in their applications. Our future work is oriented to developing more complex high performance computing (HPC) scientific application which will require powerful distributed computing resources. If needed, some modifications of the existing model will be made in the future.

References

[1] PCMag. Definition of scientific application. [Online]. Available: http://www.pcmag.com/encyclopedia/term/50872/scientific-application [retrieved: Mar., 2015]

[2] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on, Dec 2009, pp. 4–16. [Online]. Available: http://dx.doi.org/10.1109/I-SPAN.2009.150

[3] IEEE standard glossary of software engineering terminology. The Institute of Electrical and Electronics Ehgineers, New York, NY, USA. [Online]. Available: http://www.idi.ntnu.no/grupper/su/publ/ese/ieee-se-glossary-610.12-1990.pdf. [retrieved: Mar., 2015]

[4] B. Koteska and A. Mishev, "Software engineering practices and principles to increase quality of scientific applications," in ICT Innovations 2012, ser. Advances in Intelligent Systems and Computing, S. Markovski and M. Gusev, Eds., vol. 207. Springer Berlin Heidelberg, 2013, pp. 245–254. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37169-1_24

[5] R. Baxter, "Software engineering is software engineering," in Proceedings of the First International Workshop on Software Engineering for High Performance Computing System Application. Edinburgh, Scotland, United Kingdom: IEE, 2004, pp. 14–18. [Online]. Available: http://dx.doi.org/10.1049/ic:20040411

[6] C. Roy, "Practical software engineering strategies for scientific computing," in Proceedings of the 19th AIAA Computational Fluid Dynamics Conference. Red Hook, NY, USA: Curran Associates, Inc, 2009, pp. 1473–1485. [Online]. Available: http://dx.doi.org/10.2514/6.2009-3997

 ISBN: 978-1-61208-449-7

[7] D. E. Post and R. P. Kendall, “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, vol. 18, no. 4, 2004, pp. 399–416. [Online]. Available: http://dx.doi.org/10.1177/1094342004048534

[8] F. Hernández, P. Bangalore, and K. Reilly, “Automating the development of scientific applications using domain-specific modeling,” in Proceedings of the second international workshop on Software engineering for high performance computing system applications. New York, NY, USA: ACM, 2005, pp. 50–54. [Online]. Available: http://dx.doi.org/10.1145/1145319.1145334

[9] V. R. Basili et al., “Understanding the high performance computing community: A software engineer’s perspective,” IEEE Software, vol. 25, no. 4, 2008, pp. 29–36. [Online]. Available: http://dx.doi.org/10.1109/MS.2008.103

[10] L. Christopherson, R. Idaszak, and S. Ahalt. Developing Scientific Software through the Open Community Engagement Process . [Online]. Available: http://dx.doi.org/10.6084/m9.figshare.790723 [retrieved: Mar., 2015]

[11] O. Marques and T. Drummond, “Building a software infrastructure for computational science applications: lessons and solutions,” in Proceedings of the second international workshop on Software engineering for high performance computing system applications. New York, NY, USA: ACM, 2005, pp. 40–44. [Online]. Available: http://dx.doi.org/10.1145/1145319.1145332

[12] G. Wilson et al. Best Practices for Scientific Computing. [Online]. Available: http://arxiv.org/abs/1210.0530 [retrieved: Mar., 2015]

[13] J. Segal. Models of scientific software development. [Online]. Available: http://oro.open.ac.uk/17673/1/SegalICSE08R.pdf [retrieved: Mar., 2015]

[14] ——, “Scientists and software engineers: A tale of two cultures,” in Proceedings of the Psychology of Programming Interest Group. UK: University of Lancaster, 2008, pp. 44–51. [Online]. Available: http://www.ppig.org/papers/20th-segal.pdf

[15] R. Sanders and D. Kelly, “The Challenge of Testing Scientific Software,” in Proceedings of the Conference for the Association for Software Testing, July 2008, pp. 30–36. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.464.7432&rep=rep1&type=pdf

[16] I. Sommerville, Software Engineering, 9th ed. Harlow, England: Addison-Wesley, 2010.

[17] OpenMPI. Open MPI: Open source high performance computing. [Online]. Available: http://www.open-mpi.org/ [retrieved: Mar., 2015]

[18] J. C. Light and T. Carrington Jr, “Discrete-variable representations and their utilization,” Advances in Chemical Physics, vol. 114, 2000, pp. 263–310. [Online]. Available: http://dx.doi.org/10.1002/9780470141731.ch4

[19] Cutest: C unit testing framework. [Online]. Available: http://cutest.sourceforge.net/ [retrieved: Mar., 2015]

 ISBN: 978-1-61208-449-7