A New Perspective in Scientific Software Development

Atif Farid Mohammad
School of Computing, Queen's University, Kingston, ON
atif@cs.queensu.ca

Abstract: *Scientific software development is a process whereby software is created to assist scientists in achieving their target solutions. This process lacks harmonious communication between scientists and software engineers and thus, a gap needs to be breached between the scientific community and the computing world. This vital issue can be resolved by utilizing a new perspective in scientific software development, using well-established practices of software engineering. This new paradigm is discussed in a case study with several scientists, who confirm its effectiveness for developing scientific software if it can be adapted to their environment.*

Keywords
Science, software, design, documentation, techniques, tools, methodologies

1. Introduction

Computing is a branch of mathematics, and human-computer interaction is the main area providing a bridge between the computing and scientific communities. Software engineering is one of the pillars of this bridge. This paper presents an understanding of the field of scientific software development by software engineers.

Scientific software development has played a key role in the information age - a role that is expected to gain more advancement in the future. On the basis of a detailed case study, this paper will describe the process of identifying requirements of the objects and their relevant processes classification; as per the design theorist Horst Rittel [1] "*a statement of a problem is a statement of the solution.*" Even though this is a relatively uncomplicated observation, it offers a fine model often ignored by software solution providers: requirements classification for scientific software is in a premature stage due to the closed nature of the scientific world of research.

An elementary requirement of any scientific and/or engineering field is that its rational underpinnings be original and well thought out. These originations of ideas or algorithms provide initial objects and related states of these objects, which are alterable due to associated processes. This methodology is called (OPM) Object-Process Methodology, which is a combination of associated language and diagrams to depict a software model in a generalized way for everyone to understand easily.

This paper also brings forward the best practices for software engineers to adapt while working with scientists. A list has been compiled after careful review of the software engineering practices utilized in general for software development:

Table 1

A	Initial functionality configuration (Requirements)
B	Establishment of a relationship with point A
C	Establishment of a common testing ground
D	Daily log maintenance
E	Modules development tracking
F	Module inspections
G	Disaster recovery plan
H	Improvement of solution safety and security
I	Reliability
J	Code quality check

These practices arose as a result of trying to organize and categorize the interactions of the scientists [10] and software engineers' [11] work ethics. This paper is arranged in ten sections and contains detailed discussion on these practices and a case study with empirical analysis of the new paradigm.

2. Scientific Software Development

Scientific software development can be divided into five important factors: **Ideas, Concepts, Structures, Architectures** and **Operations**. Galileo had discovered that the sun was the centre of our known universe. This idea was established by Galileo on the basis of his study of Copernicus's work. The discoveries of Newton offered mathematical explanations as to why planets orbited the sun and also suggested that the whole scientific world could be explained by fundamental laws of nature. Scientific R&D contains specific relevance to accuracy and performance of research in progress [3]. Scientific R&D involves the following phases:

- Planning of research
- Organization of research ideas in logical sequence and System Study of control points
- System Development
- Individual module inspections
- Field trials and experiments

These phases require that research scientists offer guidance to software engineers involved in the development of scientific solutions. Meyersdorf, Dori et al. [3] used Object-Process Analysis to establish a sound methodology for R&D performance evaluation. Every system has a structured architecture. Scientific software architecture [5] is the embodiment of

T. Sobh (ed.), *Innovations and Advances in Computer Sciences and Engineering*,
DOI 10.1007/978-90-481-3658-2_23, © Springer Science+Business Media B.V. 2010

scientifically established concepts. For example, a change of voltage is proportional to an electric current, or a change in voltage can also be proportional to a charge to a rechargeable battery. As per IEEE 1471-2000, *software architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.* Scientific software is either built by a scientist or most efficiently by a team of scientists and software engineers. To develop a scientific software solution requirement, one needs the following:

- Conceptual modeling
- Well-defined process
- Power tools (Computer language(s))

Scientists perform several sorts of tests on their daily experiments, and extract data. This type of data processing is in fact a conceptual modeling [5] [6] performed by scientists. *In a few systems, the conceptual view plays a primary role. The module, execution, and code views are defined indirectly via rules or conventions, and the conceptual view is used to specify the software architecture of a system, perhaps with some attributes to guide the mapping of other views* [9]. It is important that a software engineer use the mix of software engineering practices to prepare the software for delivery to a scientist with an eye to upcoming changes.

3. Best Practices

3.1 An **initial functionality configuration** with an individual scientist's perspective needs to be created to communicate a shared understanding of the new product among all project stakeholders, such as scientists, software designers, and developers, etc.

3.2 A **relationship should be established** among all requirements provided by the scientists, according to the mentioned functionalities in an analysis to achieve results with an operation's prime designer's authority to authorize any changes if needed at the time of and after solution development.

3.3 A common testing ground needs to be established by all stakeholders on a modular basis to achieve an error-free solution. The solution testing should have classifications for each module in the desired software as per usage priority.

3.4 A **daily log** should be maintained by software engineers to have a back track to the actual system development as well as alterations, additions, and updates in the requirements document's deviations needed by the scientists.

3.5 All **modules development tracking** should be maintained in a development log, to manage the relationships established in the raw detail manual showing all stakeholders' mentioned requirements in the document of initial configurations.

3.6 Software engineers need to ensure that **inspections are performed by the scientists** at the completion of each module to determine the correct version on the basis of the first four points.

3.7 A solution **development disaster recovery** plan is to be injected, ensuring that versions of the solution are secured off site until the actual solution is ready to replace those in the backup repositories.

3.8 At the time the software modules are compiled, software engineers should plan to **improve safety and security** of the software through the integration of sound software engineering principles and standards to effectively meet the objectives of the solution.

3.9 Reliability is a key factor to be measured by scientists in their peer-to-peer work with software engineers, as per software reliability standards documented in NASA-STD-8739.8. "*Trending reliability tracks the failure data produced by the software system to develop a reliability operational profile of the system over a specified time.*"

3.10 Independent module **code quality checking** of the software is needed at every step of scientific software development to ensure that the modules are readable with step-by-step technical documentation to facilitate future enhancements. Software module failures may be due to some of the following factors:

- Incorrect logic
- Incorrect statements or incorrect input data

Andrew Tanenbaum has said [15]: *"The nice thing about standards is that you have so many to choose from; further, if you do not like any of them, you can just wait for next year's model."* There can be at least three major directions concluded after going through the best practices mentioned above:

I. Perspective of the scientist
II. Perspective of the user
III. Perspective of the "interface designer"

4. Design and implementation

Scientific software architecture needs to be designed by software engineers and developers such that scientists can take advantage of emerging computational environments such as the Web and information grids, and pattern identifier applications on the available data with user-friendly front-end inputs. By defining patterns in scientific software, authors [12] aspired to the development of software by the use of object components only. Whereas scientists mostly tend to use programming languages, such as FORTRAN, operations organized as toolkits to provide resultant desired solutions and association rule mining capabilities can be produced using other languages as given in [12 & 13], such as C/C++,

Python and most recently C Sharp and Java. Authors [14] mentioned the non-availability of modern aspects of software design in Fortran 90 without discussing the aspects of a certain generation of language.

5. Case Study

A case study has been prepared and conducted from actual interviews with scientists to analyze the new paradigm's affective impacts on scientific software development. Following is the questionnaire that resulted from actual interviews of scientists.

Q. 1: What will you do if you need software to help you as an integral part in achieving your goal? Please select your priorities by renumbering from selections given in Section 1 from A to J.

Q. 2: Please rate your choices in importance from 1 to 7 of the above given choices, 1 being the least important and 7 being of maximum importance.

Table 2

A		B		C		D		E	
F		G		H		I		J	

Q. 3: Do you agree or disagree with the following statement: if a software engineer and an operational user are involved in helping you achieve your goal, (please add remarks in a couple of sentences, if you like):

It is most essential for the software engineer to write down requirements of the scientist in some accepted, structured format specified by the scientist verbatim, to gather and analyze them in order to get a solution provision.

Q. 4: Will you want to inspect your required solution at each step of verification and validation by yourself or would you prefer the software code be inspected by outside consultants of the same trade as your own?

Q. 5: Please rate software security as being critical to your desired solution on the same scale of 1 to 7, where 1 is least important and 7 is of maximum importance, and 9 is used for not applicable. Please explain your rating in a few words.

Q. 6: In your opinion, is the following perception correct: that scientific knowledge has to be implemented in mathematical form? As the pen became the routine tool, so does the computer, as our technology today, demand an even stronger definition of scientific knowledge. This knowledge becomes useful only when represented by a software application. You can answer this on the scale of 1 to 7, with 1 as agreeing least and 7 as agreeing most.

Q. 7: If you ever have developed software, what problems did you face in development, and how did the process you use help you in a unique way?

Q. 8: If you involve a software engineer to get your scientific software developed, is there a unique reason? Do you prefer using the sequence of methodologies you selected in Question 1?

6. Lessons Learned

The interviews with our scientists resulted in a particularly rich resource of information about scientific software. The case study of scientific software development reported herein may not be representative of all scientists.

It is possible that scientists have provided information in the interviews regarding what they require to get their software job done, and what steps they actually would like to take, if given a chance to get it developed by themselves. Table 1 contains coding of prescribed practices in Section 1 for the interviewees, of a software engineer with scientists involved in scientific software development.

Table 3 contains coding of best practices. These codes are utilized in pictorial diagrams in bar charts.

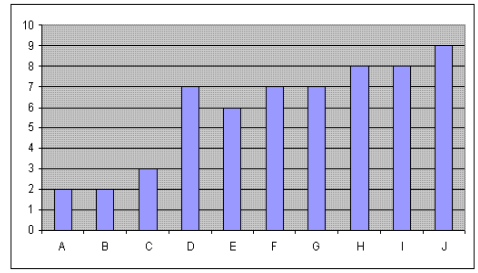
Table 3: Participant Scientists

1	BPY	2	BKK	3	BSF	4	BZL	5	BJY
6	ZAN	7	BKB	8	MP1	9	MP2	10	SP1
11	SP2								

Data has been categorized as shown in Tables 3 and 4 for the purpose of analysis. All tables contain actual data by participants. The section of lessons learned on analysis is presented to prove the effectiveness of prescribed software engineering practices toward scientific software development.

Analysis of Question 1: The selection made by participants on average for all practices generated results displayed in Chart 1:

Chart 1: Average selection by participant



The result of Question 1 clearly demonstrates the interest of scientists in the prescribed best practices and gives a new sequence shown below:

Table 4: Scientist-selected sequence of work

A	Initial functionality configuration
B	Relationship among requirements
C	Testing ground
E	Module development tracking
F	Module inspection
G	Disaster recovery plan
D	Daily log
H	Safety
I	Reliability checking
J	Code quality check

As per our participant BKK of the Department of Biology at Queen's University, the actual sequence is in perfect logical order. Selection results do not demonstrate a significant difference in the sequence.

Analysis of Question 2: This analysis brought us an equal importance in response from the scientists of the steps in the sequence from most important to least important.

Chart 2: Pictorial representation of importance

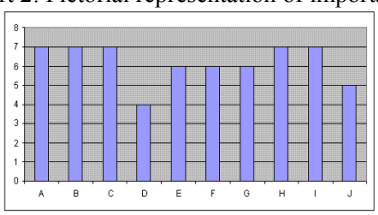


Table 5.1: Selections by participants

A	Initial functionality configuration – Most Imp.
B	Relationship among requirements – Most Imp.
C	Testing ground – Most Imp.
H	Safety – 2nd Most Imp.
I	Reliability checking – 2nd Most Imp.
E	Modules development tracking – 3rd Most Imp.
F	Module inspection – 3rd Most Imp.
G	Disaster recovery plan – 3rd Most Imp.
J	Code quality check – 4th
D	Daily log – 5th

Analysis of Question 3: In response to this question, all participants agreed with the statement. In the opinion of our expert scientist BPY: "*It is important that all stakeholders (Scientists, software engineers and operational users) are to be at one platform. This agreement will start a scientific software development process to get the objective achieved to work without any software bursts in between scientific experiments, data input and analysis.* "

Our second expert BKK said: "*Verbatim documentation might not be possible by software engineers, as scientists might come up with different ways to solve one issue. It is important to make a daily log of discussions by all participants.*"

Analysis of Question 4: Except for BKK, SP1 of York University and SP2 of Toronto College of Technology, all participants were interested in inspecting software development by themselves.

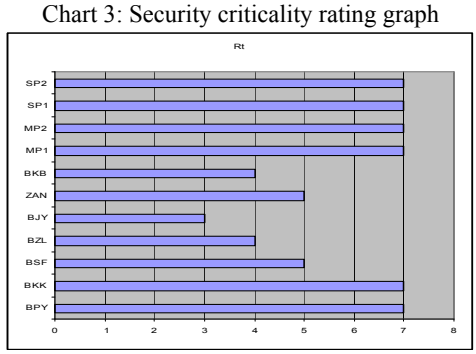
In the opinion of BKK, it will give him an edge to inspect software after getting a second opinion from industry experts of his trade to garner more comments that will help him rethink the way he finds target software for his test results.

In the opinion of SP1 of York University, outside inspectors will take up his time asking for details, which will not be that fruitful. As he is the expert, it is better that he by himself do the inspection. SP2's thoughts about software inspection, as well as software safety and security, are given below:

"*Software inspection is the progression used to help categorize the correctness, completeness, security, and value of developed software. Inspection is a procedure of technical investigation, performed on behalf of stakeholders by either outside consultants or software initializing scientists, which is intended to reveal quality-related information about the desired software with respect to the context in which it is intended to operate.*"

Analysis of Question 5: The result came as six participants out of 11 thought the safety and security to their desired software is most critical as shown in Chart 3.

Chart 3: Security criticality rating graph



As per a few participants, the comments are given below:

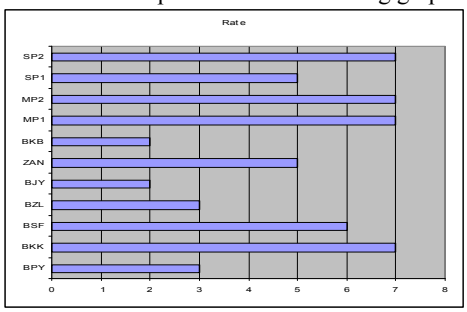
Participant BPY: *"I think scientific software developed for a scientist does not need any significant security. Scientist's software is a part of the whole experimentation process. This process might need software to only calculate some parameters, for example, to get the change in strength of a chemical's density only."*

Participant BSF: "*Software is highly valuable; it must be protected from unauthorized use. The importance of software safety and piracy continues to increase globally as our research work is now done around the world.*"

Participant BJY: *"If software is developed for any of my research work, I do not think that needs any security."*
Participant BKK: *"In any case, security is the most vital aspect to our software. It does not matter if it is commonly available in the market or if it is been developed by a software engineer."*

Analysis of Question 6:

Chart 4: Perception correctness rating graph



Analysis of Question 7: Except SP1 and SP2, all other participants have used available software in their trades. The first two mentioned participants did develop some of their desired software and commonly used the following path:

1. Conceived an idea of scientific experiment.
2. Noted all possible solutions.
3. Devised result calculation algorithms.
4. Designed a software model in UML/BPEL.
5. Coded software in C++/Java.
6. Did software testing module by module.
7. Packaged the solution for future use.

Analysis of Question 8: BPY and BKK involved software engineers and have given their thoughts as following: BPY: *I have worked with School of Computing students and faculty members to get my required software developed. Scientific software development can be considered difficult work for a software developer with little or no knowledge of the scientific curriculum. We have to explain all requirements to software engineers step by step.*
BKK: *I like to work on an idea to convert it to a mathematical model first. If I have to involve a software developer, I always try to explain in detail my devised models. I also like to show the results I get on paper after the dry run of my own algorithms. This gives an understanding to the software developer, as to what I need in actuality.*

7. Use of Methodologies

Our statistical case study and analysis shows that scientists think, innovate, and build a scientific solution to answer a specific need or to test some of their critical/innovative ideas. The case study also shows that the need for desired scientific software can usually be presented in the requirements document. There are several software development methodologies available, such as Waterfall, Spiral, Capability Maturity Model Integration (CMMI), and Object-Process Methodology (OPM). This paper will look at OPM as a formal standard to software engineering like Unified Modeling Language (UML) [02]. In reality, when we start designing a system, in general, we usually do not start from scratch. This is inevitable in the case of scientific software development, because we might have to design and develop systems of greater complexity.

8. Expanding Knowledge Base

As per our discussions on Questions 7 and 8 in the case study, the requirements that evolve in the development of a scientific solution from a possible knowledge base necessitates an understanding of ideas in a compact and easy way. In order to better comprehend these requirements, elicitation techniques are used to work with scientists. To explain why these techniques are important for the success of the products, a brief review is given below [7, 8].
Interviewing [8] is a simple, direct technique used in order to bring to the surface new information or to uncover conflicts or complexities. **Brainstorming** can be done by software engineers and scientists to define issues and questions regarding the scientific solution. **Prototyping** is an initial versioning of a system, which is available early in the development phase. **Questionnaires** are the collections of closed and open-ended questions to be provided to scientists; their responses are analyzed to understand the general trends and their opinions. **Observation** is an ethnographic technique in which software engineers can observe their design's dry-run activities as performed by scientists. **Protocol Analysis** involves the users engaging in a task and discussing their thoughts with the software engineer.

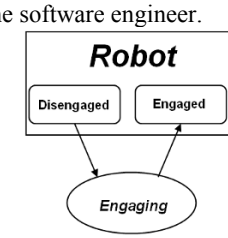


Figure 1: An example of an Object-Process diagram.

Hence, we need a visual modeling language that will allow us to compactly understand systems, incorporate parts of previously designed systems in our new designs and validate that those parts were used

appropriately. OPM [2] provides a medium to view a solution's model in a generalized way. OPM is a combination of objects, and processes that can change the states of these objects.

Objects are things in existence; process is a thing that transforms an object, and state is the situation of an object. OPM encompasses the third major factor of modeling named "State" [4]. It is a situation that an object can be in at some point during its lifecycle. At any point in time, an object is in exactly one state. A state change can happen only as a result of an occurrence of a process. A simple example toward understanding OPM is mentioned in Figure 1. A robot can have two states interchangeable due to the process of engaging.

9. Conclusion

Scientific software development is a field currently lacking an established curriculum like project management or software engineering's well-defined architectures possess. The finding in this article clearly put an emphasis on the scientists, their needs and their goals, and the prescribed way of software engineering practices to achieve workable scientific software. Scientists have ideas with the capability to convert into well thought out and proven concepts. The need to achieve their goals does exist in their minds. This requirement is primarily outside of the software engineering activity and is expressed often in fuzzy or unclear terms. Object-Process Methodology is an approach that as yet has not been properly explored in the software engineering or scientific world of research and development. OPM provides an excellent view of a system model. It can be quite suitable for scientific software modeling and development.

Acknowledgements

I like to thank Dr. Diane Kelly of Royal Military College of Canada for her wise guidance during this research work. I also like to thank Dr. Daniel Amyot, Dr. Craig Kuziemsky and Dr. Liam Peyton of University of Ottawa, as well as Dr. Paul Young and Dr. Kenton Ko of Queen's University for their invaluable input in this research work.

References

[01] Rittel, Horst and Melvin Webber (1973) "Dilemmas in a General Theory of Planning," Policy Sciences 4, Elsevier Scientific Publishing, pp. 155-159.

[02] Dov Dori, Object-Process Analysis of Computer Integrated Manufacturing Documentation and Inspection. International Journal of Computer Integrated Manufacturing, 9, 5, pp. 339-353, 1996.

[03] Mor Peleg and Dov Dori, The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods. IEEE Transaction on Software Engineering, 26, 8, pp. 742-759, 2000.

[04] Pnina Soffer, Boaz Golany, Dov Dori and Yair Wand, Modeling Off-the-Shelf Information Systems Requirements: An Ontological Approach. Requirements Engineering, 6, pp. 183-199, 2001.

[05] Liu, H., and Gluch, D. 2004. Conceptual modeling with the object-process methodology in software architecture. J. of Computing in Small Colleges, 19 (3), 10-21.

[06] Dori D, Choder M (2007) Conceptual Modeling in Systems Biology Fosters Empirical Findings: The mRNA Lifecycle. PLoS ONE 2(9): e872. doi:10.1371/journal.pone.0000872

[07] Hickey, A. Davis, and D. Kaiser, "Requirements Elicitation Techniques: Analyzing the Gap between Technology Availability and Technology Use," Comparative Technology Transfer and Society Journal (CTTS), 1 (3), pp. 279-302, 2003.

[08] Davis, O. Dieste, A. Hickey, N. Juristo, and A. Moreno, "Systematic Review of the Effectiveness of Requirements Elicitation Techniques," Proceedings of the Fourteenth International Requirements Engineering Conference (RE06), September 2006.

[09] C. Hofmeister, R. Nord and D. Soni, Applied Software Architecture, Addison-Wesley, 2000

[10] Dorian Arnold and Jack Dongarra, "Developing an Architecture to Support the Implementation and Development of Scientific Computing Applications," in The Architecture of Scientific Software, (IFIP TC2/WG2.5), Ottawa, Canada, October 2000.

[11] Ian Foster, Carl Kesselman, "Scaling System-Level Science: Scientific Exploration and IT Implications", November 2006.

[12] Charles Blilie, "Patterns in Scientific Software: An Introduction", Computing in Science and Engineering, May/June 2002, pp. 48-53

[13] Viktor K. Decyke, Charles D. Norton, Hanry J. Gardner, "Why Fortran?" Computing in Science and Engineering, July/August 2007, pp. 68-71

[14] Charles D. Norton, Viktor K. Decyke, Boleslaw Szymanski, Hanry J. Gardner, "The Transition and Adoption of Modern Programming Concepts for Scientific Computing in Fortran", Scientific Programming, Vol. 15, no. 1, spring 2007, 27 pages

[15] Tanenbaum, A.S.: Distributed Operating Systems, Prentice Hall, Upper Saddle River, NJ U.S.: Prentice Hall, 614 pages, 1995.