Development of Scientific Software and Practices for Software Development: A Systematic Literature Review

Gerardo Cerda Neumann*[1], Héctor Antillanca Espina[2], Víctor Parada Daza[3]

[1] Faculty of Engineering and Business. UCINF University. Pedro de Valdivia Avenue 450, Providencia, Santiago, Chile.

[2] Department of Computer Engineering. University of Santiago, Chile. Ecuador Avenue 3659, Estación Central, Santiago, Chile.

[3] Departamento de Ingeniería Informática. University of Santiago, Chile. Ecuador Avenue 3659, Estación Central, Santiago, Chile.

* Corresponding author. Email: gcerda@ucinf.cl, hector.antillanca@usach.cl, victor.parada@usach.cl

Manuscript submitted October 7, 2016; accepted December 12, 2016.
doi: 10.17706/jsw.12.2.114-124

Abstract: The development of adequate scientific software within the framework of a research project plays a key role in the success of the research itself. However, not all research teams complete the development of a specific software within the deadlines and with the necessary quality standards. These difficulties have been studied for a lot of years and we can conclude that these applications are difficult to create because defining their requirements is a complex task and usually developers are not fully skilled in construction of these. The following systematic literature review characterizes this kind of development and exposes its difficulties. A number of methodological solutions as well as a series of widespread practices are presented, aimed at improving the development of such kinds of software. A combination of elements is offered which would allow software development teams to choose the most adequate solution according to their specific requirements. The research results show the difference between engineering and scientific disciplines from the type of problems that it solves. Because of this, it is difficult to use a software development technique to work well in both cases and for this reason, the proposals are very useful for the future of scientific software development.

Key words: Scientific software development, software development practice.

1. Introduction

The development of scientific or research software is of fundamental importance for the success of projects that require them. It is understood that a scientific software is an application constructed within the framework of scientific research [1].

The difficulties related to the construction of this kind of software have been studied for more than ten years [2], [3]. It can be concluded that these applications are difficult to create because defining their requirements is a complex task, and in general, the developers are not necessarily experts in their construction. Although there are practices in the field of Software Engineering that can be applied as help, they have been little adopted for the construction of scientific software [4], in part because the scientists who lead the research do not have formal training in those matters.

The main activities related to the development of scientific software can be summarized from the analysis of the stages of a complex systems modeling simulation called CoSMoS [5]. Figure 1 shows the development stages as ellipses and the related products as rectangles. The solid arrows correspond to interactions and the dotted arrows indicate the creation of documents. In the figure, it is seen that the scientific software is developed from the discovery that takes place after the development. This software development in turn supports the exploration. It should be noted that the exploration is within a research context the same as the discovery, and it generates a results model. The discovery updates the domain model, and the software development is made on a work platform.

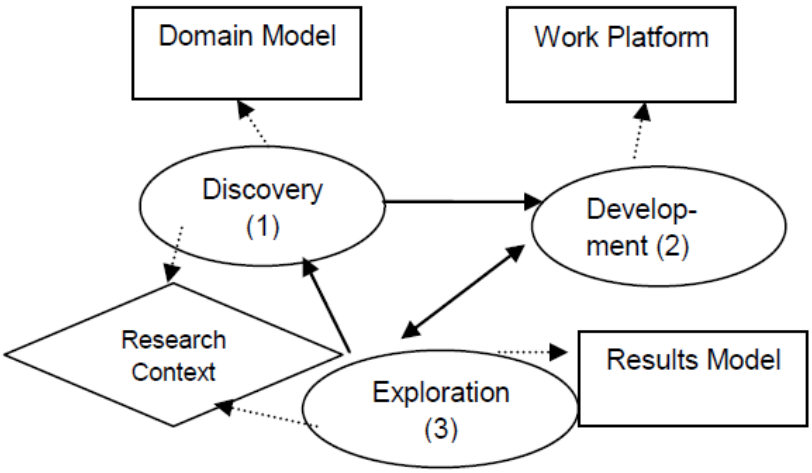

Fig. 1. Main stages in the creation of scientific software, based on [5].

In recent years several proposals meant to improve the development of scientific software have appeared. Some of them suggest the use of agile processes, i.e., those that privilege the creation of functional code over complying with formal stages [1], [6]-[9]. However, a knowledge gap persists between the scientists who head the projects and the software engineers who participate in them, hindering the use of those methods. As an example we can mention how complicated it is to implement the test-driven development (TDD), because it requires generating multiple revisions of the code as it is being written [9]. On the other hand, the increased complexity of the modeled problems that require scientific software demands high performance computational abilities, dedicated platforms, and sometimes the possibility of incorporating special hardware and specific software parts [10].

The growing complexity of the problems that scientists deal with adds a further difficulty to the development of the software constructed to solve them. A complexity factor inherent to these developments is due to the fact that the requirements are changing as the degree of understanding of the science that provides the project's context varies [11]. Consequently, the creation of software in scientific research is increasingly relevant [12], and therefore the methodology used for its development and in its tests becomes more important [1].

Many of the tools belonging to Software Engineering used in the development of traditional applications are often not useful for the construction of scientific software. Because of this, the concept of *Computational Science and Engineering* (CSE) [13], which defines the context of this type of development, is introduced. CSE tries to apply properly Software Engineering tools in the scientific research projects. In fact, the authors point out the publication of three special issues on this matter [14], [15], [16]. These kinds of efforts allow avoiding dramatic experiences, such as retracting on the knowledge published in research journals due to the faulty construction of the software detected after finishing the project [1], [17].

The present paper analyzes the development characteristics of scientific software, its main difficulties,

and methodological proposals are made to overcome them. Some conclusions of the study are also made, and two future lines of work are proposed

2. Methodology

The systematic review was based on three important sources of information: IEEE, Science Direct, and Springer. In all these cases use was made of the following string: "*scientific software development or software development practice*." Then a filter was used according to the year (2013 to 2016) and the type of publication, considering conferences and journals. It was also required that the documents should be *full text* and written in English. This led to the following breakdown:

1) 18 articles from IEEE
2) 41 articles from Science Direct
3) 15 articles from Springer.

Then the results were arranged according to relevance, and they were reviewed one by one, selecting only the ten first of each list that had content relevant to the research. To the articles selected by the search, 13 other references were added that were considered useful and were included in the reviewed papers.

Then they were classified according to their subjects, and the common matters were analyzed to extract useful elements for writing the article.

Finally, these matters were arranged, forming the coherent product that is presented.

3. Characteristics of the Development of Scientific Software

From the main studies reviewed, the following characteristics of the development of scientific software stood out [18]-[20]:

1) The software is oriented at supporting the scientific question.
2) It is developed by experts in the studied matter.
3) It had to be designed before finishing the research.

These three characteristics are analyzed in detail below.

The software must be constructed to answer the question that gives origin to the research. This means that at the beginning of the project there is only a very imprecise idea of the requirements of the software to be constructed. This perception changes as the research progresses, and consequently when the research group goes deeper into the studied subjects. Furthermore, although the construction of the software is not necessarily the main reason for the research, it is usually an important aspect of it.

The second characteristic of the development of the scientific software is that it is created by experts in the subject that is being investigated, i.e., by members of the team. Therefore, unless a member of the team is a software engineer, the construction of the applications falls on persons who do not necessarily manage the proper techniques. On the other hand, the incorporation of engineers in the research group requires them to become immersed in the issues that are being studied [12]. Both situations cause difficulties in the creation of software pieces that fulfill the requirements and have an acceptable quality. Obviously, if in a research project there are only engineers, it may be very difficult to face developments that go beyond a certain level of complexity as a result of the subject that is being studied. It is also mentioned that few scientists have the technical knowledge to carry out the testing of the software correctly [12].

The third characteristic comes from the uncertainty with respect to the result that would be obtained with the developed application. This is important if it is considered that, it is difficult to force the evolution of the project, as well as to calculate precisely the number of developers needed and the skills that they must manage. Some authors [21], [4] analyze two options to overcome this uncertainty: use either a prescriptive framework of the Software Process Improvement (SPI) that defines the specific practices and their priorities for achieving a quality level, or else use an SPI support that leaves the selection of the

process improvements to the research team.

In practice, it is better to use a hybrid decision model based on the experience of the team in the domain of the research subject. This proposal makes sense only when dealing with projects carried out by teams that already have a common experience. In spite of this, it is interesting to analyze the three research questions that must be answered to apply this Scientific Software Process Improvement Framework, SciSPIF [21]:

- What conventional planning criteria are applicable to the scientific software development projects?
- What other factors affect the priorities and decisions during the planning activities of scientific software development processes?
- What planning criteria of the scientific software development process apply to the decisions on the specific practices of Software Engineering?

These questions give an idea of the challenges of developing this type of software. Anyway, the above authors explain that until more data are available, it cannot clarify the SciSPIF decisions. Now, unfortunately, this framework is not ready for the scientific community utilizes it as a resource [4]. Several of the problematic characteristics of the development of scientific software can be approached by the *agile methodologies* [9], which have many common characteristics with the construction of the scientific software. In fact, several agile practices support the development of this type of software, as shown in Table 1.

Table 1. Comparative Agile Development / Scientific Software Development

Agile development	**Characteristics of scientific software development**
Modularity of activity development	1. Scientists and developers have some activities in the software development process.
Iteration with cycles	2. Scientists and developers can modify both the model and the code when unexpected results appear.
Limited time with iterations	3. Since the scientists need to execute the experiment as soon as possible, work is done with very limited and defined time in each iteration.
Parsimony	4. Scientists and developers do not execute unnecessary activities due to the time restrictions.
Adaptive	5. Scientific software tends to evolve due to the new results obtained in the experiments, the development of models, and the continuous increase of the functionality.
Incremental	6. New characteristics coming from new findings may be added to the application. These new characteristics are added in small steps.
Convergence	7. The developers do not know the details of all the requirements, since they write the code as needed.
People-oriented	8. Strict planning may not be adequate for the development of the scientific software because the researchers need flexibility to carry out the experiment as fast as possible.
Collaborative	9. The developers can be supported by the scientists to understand well the problem that is being studied.

From Table 1 it is possible to identify the main complexities of the creation of the scientific software:

i. Urgency in executing the experiments with the developed software.

ii. Evolution of the software as the research progresses.

iii. Difficulty for planning the activities due to the changing nature of the research.

These challenges are commented below.

Executing software in a research project, an activity called experiment, involves coding qualitative data and narratives in theoretical constructs and defining the relations between them [22]. In this way, the qualitative information can be analyzed systematically to generate new theories based on real world data, adding even more complexity to these kinds of developments [4]. Managing this evolution is not trivial, especially if one must plan the software development activities as one goes deeper into the topics belonging to the research.

The characteristics presented in Table 1 allow identifying the main difficulty to be solved: how to set up a research team such that there is sufficient coordination to construct the scientific software at the required time with an acceptable quality.

Although the difficulty to form an appropriate development team will also be present in the development of the commercial software, it is aggravated due to the complexity of the domain of the problem that is being studied and the probable lack of software engineering knowledge by the developers [23]. In fact, three large steps are defined for the construction of the scientific software [24]:

1) Prepare discrete models.
2) Translate the models into algorithms.
3) Code the models using programming languages.

Due to the complexity of this activity, errors can be introduced in any of the steps [1].

In this way, the developers should have two basic characteristics to perform their job: mastery of the subject and creativity. The former allows to understand the context of the problem and what it is desired to program; the latter allows, in the words of Edward De Bono: "*breaking out of established patterns in order to look at things in a different way*" [25]. Creativity is considered as a determining factor in software engineering, and it is fundamental when finding different solution alternatives to unexpected problems is needed.

Added to the above, the question that should be asked is what challenges must be solved.

4. Main Difficulties and Challenges of the Development of Scientific Software

Answering the scientific question of the research generates a very dynamic and demanding development environment [10]. This forces the work team to have the ability to communicate correctly both in and out of the research. Furthermore, the addition of the difficulty of having to adapt new technologies and tolerate the ambiguities characteristic of the research generated during the development of the project, is also common. In this context, the role of the software developer is seen as increasingly important. This developer is seen as capable of constructing the required application by himself, since the rest of the team put their best efforts in the initiatives belonging to the research more than on the generation of the code.

Due to the characteristics of the research projects, one of the problems for the construction of software corresponds to the difficulty to define all the requirements at the beginning of the project [9].

Other problems are also identified in the literature on scientific software development projects [10]:

1) Ambiguities and misunderstandings.
2) Teams distributed in different geographic areas, causing time differences, problems for sharing knowledge, and complications for clarifying the requirements.
3) Researchers who tend to generate rapidly a code that works, but whose results are difficult to repeat [12]. This has to do with the specificity of the generated code and its apparent or real inability to extrapolate from that code a generalization applicable to other research.

In brief, the development of scientific software that is fast, reliable, and fulfills the required functionalities is needed. This represents a great challenge because the developers are not necessarily trained in software engineering. In any decision associated with the development it must be kept in mind that the research depends strongly on the construction of the software, and any fault or delay can retard getting the scientific discoveries.

5. Methodological Proposals for Improving the Development of Scientific Software

After presenting the difficulties detected in the development of scientific software, the proposals of solutions found can be presented. These solutions have been grouped on the one hand into methodological improvements referred to the different forms of construction of the software, and on the other hand into the agile practices that are feasible of incorporating in the development.

5.1. Methodological Improvements

In general, researchers do not have training in software engineering, so it is difficult for them to interact in multidisciplinary teams [6]. This has led software to be historically undervalued and disregarded in research, even though it is fundamental for its success [18]. Based on this analysis, the following suggestions are made [12]:

1) To set up Research Software Engineering (RSE) groups.
2) To incorporate basic software development training in all doctoral programs.
3) To apply policies to ensure that software is recognized as important in the researcher community.
4) To disseminate the benefits of open source.
5) To use the software of other research and give credit to the creators.

With respect to the use of *open source* applications, there are authors who suggest using their concepts in scientific areas, i.e. [6], [26]:

1) Create and share software without restriction.
2) Generate improvement of the applications in the scientific community and make them available to everyone.
3) Not having ownership of the code by anyone in particular.

These initiatives have aroused interest although they have not been consolidated yet.

The analyzed proposals also consider making use of the experience in software construction or *startup* companies (emergent companies that rely strongly on technology and need to generate rapidly applications that allow them to operate). The similarities with scientific software construction are found in the search for rapid creation in a highly changing environment where the requirements are not clear at the beginning of the project, and they can change substantially as time goes by.

The suggestions taken from these types of enterprises are [27]:

1) To create flexible teams that adapt to change the direction of the development according to the modifications of the target market.
2) To use light methodologies to achieve flexibility when adapting practices and reacting to the software changes according to the business strategies that appear.
3) To generate fast release of applications to create an evolutionary prototype with the purpose of achieving rapid feedback from the users.
4) To empower the team members to achieve great flexibility that will allow them to explore different alternative solutions autonomously.

The proposal arises for automating the data recovery and preparation process for the analysis, i.e., for performing the experiments [28]. In this case the described experience refers to an application made to measure for the U.S. Department of Energy's Atmospheric Radiation Measurement (ARM) program).

Another relevant issue is the possibility of sharing software made by specialized scientists. For example, in the matter of visual display of research data [29], to generate an open code components standard to facilitate the integration to construct solutions for the simulation [30]. An additional example would be to create a work environment for the operation and analysis of experimental data [31]. Combining characteristics of dynamic libraries with an interpreter that has image processing functions can be mentioned [32]. The CoSMoS model could also be used to validate complex simulation systems [5] or else support the data follow-up and analysis stages [33]. All these proposals coincide in that they are based on constructing and sharing with the scientific community solutions created by specialized researchers in each of the topics. This presents the challenge of coordination and support among the different research groups around the world, a fact that forces to define common standards. In fact, one of the articles describes the challenges for creating an integrated environment that allows modeling processes of the Earth's surface by

the Community Surface Dynamics Modeling System (CSCDM) [34].

Other authors propose the development of a system for gathering and using the scientific and technological results generated in the research and development [35]. There also are researchers who use software packages dedicated to processing the generated data [36][37] and even applications developed by themselves [38].

With respect to testing of the constructed applications, it should be noted that the challenges can be grouped into two main categories [1][39]:

1) Those that are produced due to the characteristics belonging to the scientific software.
2) Those related to cultural differences between the scientists and the software engineering community.

Among the former we can find those related to the development of test cases, the difficulty to determine which should be the correct outputs to be compared, the excessive times for the execution of the tests, or rounding or truncating errors. Among the latter, we can highlight those relative to the limited understanding of the testing concepts, the use of unsystematic testing techniques, or faults in the execution of the unit tests.

Finally, it should be mentioned that the processing of the data generated in the research can be facilitated by means of Cloud Computing types of applications [40]. However, this requires the researchers to acquire solid knowledge and skills in the development of sequential and High Performance Computing (HPC), something that has not happened yet.

5.2. Incorporation of Agile Practices

As a way of overcoming the difficulty of not having the requirements defined from the beginning of the project, it is proposed to use agile development to decrease the risks in these kinds of projects [9]. In fact, Table 1 presented all the characteristics of the development of scientific software, proposing an agile practice to solve it. However, the authors themselves comment that it is difficult to introduce these agile practices in the development teams. They point out that it is necessary to make a great effort to train them and motivate them. Finally, they conclude that agile practices are useful for exploratory, iterative, and collaborative developments, something characteristic of scientific research. This is confirmed by Jirotka, Lee, & Olson [7], who point out that an agile approximation facilitates the interaction between the scientists and the engineers. In fact, they mention that the use of peer programming is more useful than using software development methods guided by plans.

However, an inappropriate use of the agile methods can inhibit creativity. Trying to develop a solution without a careful monitoring framework can be a disaster. Software engineers must keep an open mind and understand their role and responsibilities with the research team.

Agility also proposes improvements for testing the constructed applications. One of the most solid and widely disseminated proposals is that called Test-Driven Development (TDD), which provides the XP, or Extreme Programing method [11]. This technique allows the generation of a test for each code line that has failed, in this way creating a more robust software from the begnning of the coding. However, it demands great skill by the programmer that uses it and the constant support of the researchers with respect to the topic that is being studied.

Another issue analyzed in the literature is related to the contribution of the agile methods to the communication problems when the project's teams are physically apart. In this context it stands out that there are occasions in which development teams are formed with the logic of "following the Sun", i.e., succeeding in having at all times team members making developments around the world [8]. With teams constituted according to this scheme, problems appear related to the temporal, geographic, and cultural distance. The following problems can be highlighted:

1) Temporal distance: Communication must take place at unconventional times due to the lack of superposition of work hours, leading to overtime. Delays can be caused when some of the researchers need to interact with others who are not available.
2) Geographic distance: "Face-to-face" meetings are difficult to carry out, and informal communication is often insufficient. This complicates the sharing of ideas.
3) Cultural distance: It generates many misunderstandings because there are very different ways of thinking and acting between the different components of the project's team.

The following is proposed to solve these difficulties [8]:

1) To assign to parts of the team the creation of the requirements documents which are then shared with the remaining components. Although this requires good coordination with and trust in others, it allows all to review what is being done, in order to detect errors and share the advances made.
2) To make the definition of the delivery of advances in two stages. In a pre-stage, the aim is to achieve a mutual understanding of what will be implemented in the later stages. The pre-stage meetings will take place at a time at which the person responsible for defining the requirements can participate. This leads to the meeting of the stages proper as a very realistic vision of what is needed.
3) To make the revisions of the closure of each software delivery with two different audiences: first with the person responsible for defining the requirements, and then with the rest of the team. This allows having the ideas of what needs to be constructed well clarified before discussing it with the whole team.

To carry out a retrospective study at the end of each important stage to detect what was done well and reinforce it, and on the other hand, to discover what was done badly to correct it. This activity is generally ignored when trying to go rapidly to the next stage, causing repetitions of the errors and inefficiencies over time. Spending some time in these retrospective studies leads to increased productivity.

6. Conclusions and Proposals

In the analysis that has been presented it can be stressed that work is done in various fronts to overcome the detected weaknesses [41]. Among them, the efforts made in the community that uses the applications to get training in their use can be mentioned. Along this line, the development and use of sophisticated frameworks such as AIBench, based on the Java programming language and has been applied mainly in biomedicine, should be mentioned [42].

It also aims to influence the policies that motivate those involved in the scientific software community so that they venture to use support solutions.

The difference between engineering and scientific disciplines starting with the nature of the problems that they solve allows understanding why the tools that work in the former do not work in the latter. Engineering refers to the social needs to be satisfied, while science deals with the relation with one or more scientific theories and the phenomena that describe them [43]. Obviously, it is difficult for a software development technique to work well in both situations.

Different authors also suggest that the agile practices analyzed are naturally useful for exploratory, iterative, and collaborative developments, something quite characteristic of scientific experiments [9]. The challenge is defined as how to inculcate software development practices in the researchers, taking into account that they were not trained in this field during their professional formation [6]. These same authors rely on changing the research and software development culture by integrating better the agile practices.

The proposals that suggest improving the testing techniques to prevent the undetected errors from seriously damaging the research are also highlighted [1].

The analysis presented indicates that the contributions for improving the development of scientific software have decanted into the two suggested lines: methodological improvements in general, and the use of agile practices in particular.

An important contribution that would be very useful is the proposal of a development method that arranges the work of the researchers and Software Engineers. This method would have to incorporate the identified agility proposals, and it must be easy to understand and apply by the researchers and all the participants in the projects.

The following activities are proposed for future work:

1) Coordinating the researchers to share pieces of software that are useful to all.
2) Developing a simple and noninvasive method that allows the research teams to coordinate better and increase productivity.

References

[1] Kanewala, U., & Bieman, J. (2014). Testing scientific software: A systematic literature review. *Information and Software Technology*, 1219-1232.

[2] Segal, J. (2004). The Nature of evidence in empirical software engineering. *Proceedings of the Annual International Workshop on Software Technology and Engineering Practice*.

[3] Segal, J. (2005). When software engineers met research scientists: A case study. *Empirical Software Engineering*, *10*, 517–536.

[4] Mesh, E. S., Burns, G., & Hawker, J. S. (2014). Leveraging expertise to support scientific software process improvement decisions. *Software Engineering for CSE*, 28-34.

[5] Polack, F. (2015). Filling gaps in simulation of complex systems: The background and motivation for CoSMoS. *Natural Computing*, 49-62.

[6] Ahalt, S., Minsker, B., Tiemann, M., Band, L., Palmer, M., Idaszak, R., *et al*. (2013). Water science software institute. *An Open Source Engagement Process*.

[7] Jirotka, M., Lee, C., & Olson, G. (2013). Supporting scientific collaboration: methods, tools and concepts. *Computer Supported Cooperative Work*.

[8] Korkala, M., & Maurerb, F. (2014). Waste identification as the means for improving communication inglobally distributed agile software development. *The Journal of Systems and Software*, 122-140.

[9] Nanthaamornphong, A., Morris, K., Damian, W. I., & Hope, A. M. (2013). A case study: Agile development in the community laser-induced incandescence modeling environment (CLiiME). *SE-CSE IEEE*, 9-18.

[10] Marinovici, C., Kirkham, H., & Glass, K. (2014). The hidden job requirements for a software engineer. *IEEE Computer Society*, 4979-4984.

[11] Nanthaamornphong, A., Carver, J., Morris, K., Michelsen, H. A. & Rouson, D. W. I. (2014). Building CLIIME via test-driven development: A case study. *Computing in Science & Engineering*.

[12] Goble, C. (2014). Better software, better research. *IEEE INTERNET Computing*.

[13] Carver, J., Epperlyy, T., Hochsteinz, L., Maxville, V., Pfahl, D., & Sillito, J. (2013). *Proceedings of the 5th International Workshop on Software Engineering for Computational Science and Engineering*.

[14] Carver, J. (2009). First international workshop on software engineering for computational science & engineering. *Computing in Science Engineering*.

[15] Carver, J. (2009). Report: The second international workshop on software engineering for CSE. *Computing in Science Engineering*.

[16] Carver, J. (2011). Software engineering for computational science and engineering. *Computing in Science & Engineering*.

[17] Miller, G. (2006). A scientist's nightmare: Software problem leads to five retractions. *Science*.

[18] Carver, J., Kendall, R., Squires, S., & Post, D. (2007). Software development environments for scientific and engineering software: A series of case studies. *Software Engineering*.

[19] Kelly, D., Thorsteinson, S., & Hook, D. (2011). Scientific software testing: Analysis with four dimension. *Software IEEE*.

[20] Nguyen-Hoan, L., Flint, S., & Sankaranarayana, R. (2010). A survey of scientific software development. *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*.

[21] Mesh, E. S., & Scott, H. J. (2013). Scientific software process improvement decisions: A proposed research strategy. *SE-CSE IEEE*.

[22] Szymczak, D., Smith, S., & Carette, J. (2016). A knowledge-based approach to scientific software development. *Proceedings of the 2016 International Workshop on Software Engineering for Science*.

[23] Zhu, X., Yurteri-Kaplan, L., Park, A., Sokol, A., Iglesia, C., Gutman, R., *et al*. (2014). An observational ergonomics job analysis method to quantify postural risk factors for musculoskeletal injury among gynecologic vaginal surgeons.

[24] Dahlgren, T., & Devanbu, P. (2005). Improving scientific software component quality through assertions. *Proceedings of the Second International Workshop on Software Engineering for High Performance Computing System Applications*.

[25] De, B. E. (1973). *Lateral Thinking: Creativity Step.* New York: Harper & Row Publisher.

[26] Silva, V., Crowley, H., Pagani, M., Monelli, D., & Pinho, R. (2014). Development of the openquake engine, the global earthquake model's open-source software for seismic risk assessment. *Nat Hazards*.

[27] Paternoster, N., Giardino, C., Unterkalmsteiner, M., Gorschek, T., & Abrahamsson, P. (2014). Software development in startup companies: A systematic mapping study. *Information and Software Technology*.

[28] Gaustad, K., Shippert, T., Ermold, B., Beus, S., Daily, J., Borsholm, A., *et al.* (2014). A scientific data processing framework for time series NetCDF data. *Environmental Modelling and Software*.

[29] Ryabinin, K., & Chuprina, S. (2013). Adaptive scientific visualization system for desktop computers and mobile devices. *Procedia Computer Science*.

[30] Tang, Z., Zhang, W., & Wu, P. (2014). A holistic framework for engineering simulation platform development gluing open-source and home-made software resources. *Advances in Engineering Software*.

[31] Pérez-Rodríguez, G., Glez-Peña, D., Azevedo, N. F., Pereira, M. O., Fdez-Riverola, F., & Lourenc, A. (2015). Enabling systematic, harmonised and large-scalebiofilms data computation: The BiofilmsExperiment Workbench. *Computer Methods and Programs in Biomedicine*.

[32] Nedzved, A., Gurevich, I., Trusova, Y., & Ablameyko, S. (2013). Software development technology with automatic configuration to classes of image processing problems. *Pattern Recognition and Image Analysis*.

[33] Mason, S. J., Cleveland, S. B., Llovet, P., Izurieta, C., & Poole, G. C. (2013). A centralized tool for managing, archiving, and serving point-in-time data in ecological research laboratories. *Environmental Modelling & Software*, pages 59-69.

[34] Peckham, S. D., Hutton, E. H., & Norris, B. (2012). A component-based approach to integrated modeling in the geosciences: The design of CSDMS. *Computers & Geosciences*.

[35] Biktimirov, M., Polikarpov, S., Scherbakov, A., Efremov, P., & Solodkin, D. (2014). The development of a system for the collection and use of scientific and technological results. *Scientific and Technical Information Processing*.

[36] Krumm, P., Zuern, C., Wurster, T., Seeger, A., Mangold, S., Klumpp, B., *et al*. (2013). Pre- and post-interventional analysis of myocardial strain in patients undergoing mitral valve clipping using cardiac *MRI. Scientific Sessions - Insights Imaging*.

[37] Lánczi, L., Balázs, E., Béres, M., Tircsó, G., & Berenyi, E. (2013). Comparative examination of magnetic resonance contrast agents on low and high magnetic field. *ESMRMB 2013 Congress*.

[38] Hansen, J., Wielpütz, M., Pahn, G., Skornitzke, S., Grenacher, L., Kauczor, H.-U., *et al*. (2015). Formation of a well-defined arterial input function for contrast-enhanced CT using a pre-determined patient-specific circulatory function for individual contrast-agent bolus-shaping. *Scientific Sessions and Late-Breaking Clinical Trials.*

[39] Kanewala, U., Bieman, J., & Ben-Hur, A. (2016). Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels. *Software Testing Verification & Reliability*.

[40] Church, P., Goscinski, A., & Lefèvre, C. (2015). Exposing HPC and sequential applications as services through the development and deployment of a SaaS cloud. *Future Generation Computer Systems*, 24-37.

[41] Crouch, S., Chue Hong, N., Hettrick, S., Jackson, M., Pawlik, A., Sufi, S., *et al*. (2013). The software sustainability institute: Changing research software attitudes and practices. *Computing in Science & Engineering*.

[42] López-Fernández, H., Reboiro-Jato, M., Glez-Peña, D., Méndez Reboredo, J. R., Santos, H. M., *et al*. (2011). Rapid development of proteomic applications with the AIBench framework. *Journal of Integrative Bioinformatics*.

[43] Angius, N. (2013). The problem of justification of empirical hypotheses in software testing. *Philosofy Technology*,

Gerardo Cerda Neumann is a professor at UCINF University and UNIACC University in Chile. He is a doctoral student at the University of Santiago, Chile. His areas of interest include are software engineering, process management and big data

Héctor Antillanca Espina is an assistant professor at the University of Santiago, Chile. He has a doctorate in Engineering Science. His areas of interest include are software engineering and CSCW.

Víctor Parada Daza is an associate professor at the University of Santiago, Chile. He has a doctorate in Engineering Science. His areas of interest include are stochastic optimization and genetic programming.