PUC Minas
A B A K Ó S
Instituto de Ciências Exatas e Informática

On the Validation of a Specific Development Process for Scientific Software using the Inspection Technique*

Jether Oliveira Gomes[1]
Gray Farias Moita[2]

Resumo

A crescente demanda por *software* tem gerado maior necessidade por processos para desenvolvimento de programas. No entanto, embora a literatura não apresente processos para o desenvolvimento de *software* científico, um Processo de Desenvolvimento Específico para *Software* Científico (SDPSS) foi proposto em estudos anteriores. SDPSS foi concebido tendo como base a Metodologia de Humphrey, a qual define a necessidade de executar o processo de aquisição em oito passos. Tendo em mente que as primeiras seis etapas já foram realizadas, este estudo considera a realização das sétima e oitava etapas da metodologia, que são a validação do processo inicial e, posteriormente, a sua melhoria. É importante destacar a importância desses passos, porque um processo deve ser testado e validado para que se possa assegurar a sua eficiência. Assim, primeiramente, uma aplicação *web* foi desenvolvida para automatizar e modelar o SDPSS de modo a garantir sua implementação. Subsequentemente, SDPSS foi aplicado em ambientes reais de desenvolvimento de *software* científico para se obter resultados que fossem analisados por meio de uma técnica de validação por inspeção, conhecida como VProclnsp. Apesar de algumas inconsistências, concluiu-se que o SDPSS pode realmente contribuir para o desenvolvimento de *software* científico acadêmico.

Palavras-chave: Processos de *Software*. Processo de Validação. Software Científico.

*Submetido em 25/02/2015 – Aceito em 30/03/2015

[1]Doutorado em Modelagem Matemática e Computacional, Programa de Pós-graduação em Modelagem Matemática e Computacional, Brasil – jethergomes@yahoo.com.br

[2]Doutor em Aeronáutica, Programa de Pós-graduação em Modelagem Matemática e Computacional, Brasil – gray@dppg.cefetmg.br

Abstract

The increasing demand for software has generated greater need for software development processes. However, although the literature does not present processes for the development of scientific software, in previous studies, a Specific Development Process for Scientific Software (SDPSS) was proposed. SDPSS was conceived based on Humphrey's Methodology, which defines that the execution of eight steps is necessary for the acquisition of a process. Having in mind that the first six steps have already been carried out, the current study considers the accomplishment of the seventh and eighth steps of the Methodology, which are the validation of the initial process and its later improvement. It is important to highlight the significance of these steps because a process must be tested and validated to ensure its efficiency. Hence, first, a webapp to automate and model the SDPSS was developed to aid in its implementation. Subsequently, SDPSS was applied in real environments of scientific software development to obtain results that were analyzed through a validation technique by inspection, known as VProcInsp. Despite some inconsistencies, it was concluded that SDPSS can really contribute to the development of academic scientific software.

Keywords: Software Processes. Process Validation. Scientific Software.

1 INTRODUCTION

With the increasing evolution of its power of storage and processing, and diminishing costs, computers are increasingly present in the society nowadays, which, in turn, increases the dependence on these machines and their software every day. The latter have presented a significant increase in its internal complexity, resulting in a greater error incidence and, consequently, a decrease in its quality (PEREIRA; MOITA, 2008).

To manage the larger internal complexity of the software, software engineering techniques have been used to allow a better control on the development process, and thus, to avoid undermining its quality. These techniques – known as Software Development Processes – when well used, allow high reliability and quality software developments. A software development process is a set of activities, partially ordered, with a purpose of obtaining a product of software. The question surrounding the software development process is mainly the stability, control, and organization for the product development activity.

Although there are many researches on Development Processes in the literature, none of them covers all kinds of projects, because usually each project has its own and different needs, and, as a consequence, the selected software process usually cannot be applied without any customization (KALUS; KUHRMANN, 2013).

Within this subject, a topic that deserves careful attention is the study of the development process for scientific software of academic nature. To accomplish this, an investigation of the specific processes for academic environment was carried out and the existent processes were verified to have their main focus on the production of conventional software (PEREIRA; MOITA, 2008). Consequently, from this finding, a new process – the so-called SDPSS (Specific Development Process for Scientific Software) – was proposed, which is described in Section 2 of the present paper. This process was conceived and defined from the first six steps of Humphrey's Methodology (HUMPHREY, 1995). In the present study, the two remaining steps of the Methodology have been employed.

The seventh step consists of validation of the initial process, while the eighth and last step comprises the correction and improvement of the process. Validation of the initial process is an important step, and can address a natural concern of project management – the guarantee that the development process guides the participants in an efficient and correct way during the creation of the software. Excessive bureaucracy or an ambiguous orientation can confuse, instead of guiding, the process of development in the product creation cycle

In this context, the present study aims to show the results of SDPSS validation using an inspection technique that aims to investigate if the new process provides effectiveness to the scientific software development of academic nature.

The following section shows some general concepts of software development processes,

especially the SDPSS, which motivated this research. Section 3 shows the processes validation, while Section 4 presents the validation of the SDPSS. Section 5 points out the results of the validation, and finally, Section 6 shows the final considerations about the validation of the SDPSS.

2 SOFTWARE DEVELOPMENT PROCESSES

In every software development process, meeting the requirements, accomplishing deadlines, adjusting real costs and resources, and complying with many other specificities demanded to guarantee good quality software are not simple tasks to be accomplished. An extremely detailed monitoring of the tasks is particularly necessary, which involves its entire construction from the creation until its installation at the final client. Accordingly, Software Engineering came up with a specific technique to guarantee quality and trustworthiness in the execution of the tasks, namely Software Development Processes: A set of systematized activities, partially ordered, which make an effort to acquire the software product.

With time, it has been proved that when the development process is focused, it can guarantee the quality of the software since the beginning of its manufacturing, because it can be controlled step-by-step and its quality can be tested before it reaches the client.

The software processes are the basis for managing control of software projects, and they establish the context in which the technical methods are applied, the working products (models, documents, data, reports, forms) are produced, the marks are established, the quality is assured, and the changes are adequately managed. Pressman (2001) highlighted that the process is the main point that keeps the union of the technology layers. Figure 1 shows the Software Engineering layers.

Even with the known benefits obtained with the use of the countless existing software development processes, the application and introduction of these techniques are still complex tasks and highly dependent on the environment in which they are placed in.

In an attempt to try to find a predictable methodology that improves productivity and quality, several models of software development process with different bases, such as, Waterfall, Iterative Process, Agile Processes, Unified Process, among others were constructed. Such processes are discussed and compared in Silva, Souza and Camargo (2013) and Purri (2009). Furthermore, there are quite a few processes that do not fulfill the expectations in all the extents and there is no model that can be regarded as a suitable process for all the applications.

In this context, an issue that deserves special attention is the study of the scientific software development process of academic nature, which is detailed in the study of the SDPSS.

Figure 1 – Software Engineering Layers [3]

Source: PRESSMAN, 2001

2.1 SDPSS

The SDPSS considers the term scientific software as “software developed by researchers in their scientific research projects". A vast amount of these projects are of scientific nature or, in other words, research projects of scientific initiation, master’s degree, doctoral and postdoctoral stages, and others, which rely on software creation to help in their researches (PURRI, 2009).

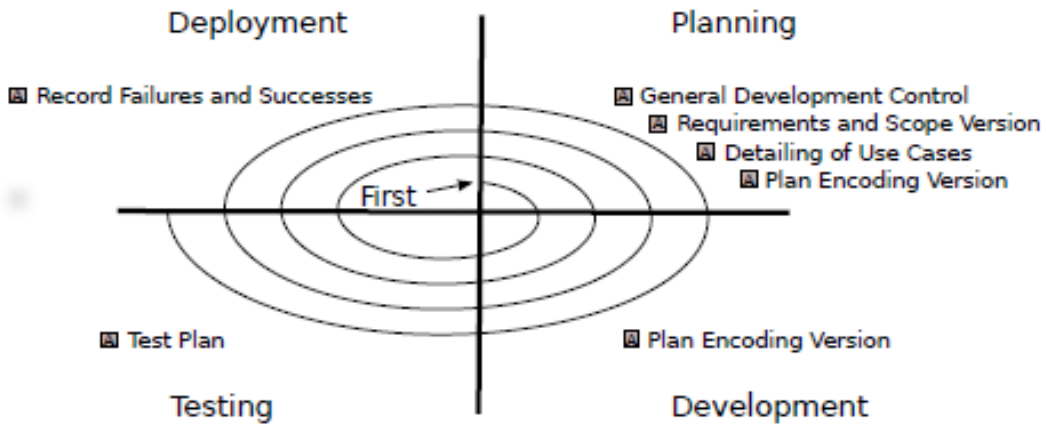
Motivated by the lack of processes for the construction of scientific software of academic nature, the SDPSS was thought as a possible alternative. It was initiated by Purri (2009) and advanced with the work of Pereira and Moita (2008) both during their M.Sc. studies at the Computing and Mathematics Modeling Program at the Federal Center for Technological Education of Minas Gerais – CEFET-MG, Brazil. As already mentioned, Humphrey’s Methodology was used for the definition of the process.

The initial approach is strongly influenced by the Unified Process (UP) and Extreme Programming (XP) methodologies. The main features that were adapted from UP are the iterative and incremental life cycle. Both SPDSS and UP are processes with a few artifacts, with a little bureaucracy and that allow fast software development. However, SDPSS is established for scientific purpose, whereas UP has a commercial approach. In relation to XP, it is possible to notice similarities in the standardization of the source code, in the simplicity of the process, in the design of small versions in each iteration of the iterative and incremental life cycle, in the continuous integration and in the collective ownership of the source code (PEREIRA; MOITA, 2008).

The life cycle used in the SDPSS is defined as iterative and gradual. In each of the iterations, the following phases are contemplated: Planning, Development, Testing, and Deploy-

ment, and consequently, the creation of its respective artifacts, as shown in the spiral depicted in Figure 2.

Figure 2 – Life cycle proposed by the SDPSS



Source: Prepared by authors

The four phases of the development can be described as follows:

- **Planning** – In this phase, each of the iterations must be planned to be small to maximize the development control. All the iterations must be designed to be simple, yet should not miss any important detail in the development.
- **Development** – Based on the fast and reduced planning made previously, a version of the iteration in development is created, observing the defined codification standards.
- **Testing** – After completion of the codification, each version must pass the integration and unit tests. The unit tests are suitable to guarantee that this version works according to the plan. After the approval of the unit tests, the integration process is run, so that the compatibility of the new code with the existing version is guaranteed.
- **Deployment** – After passing the integration and unit tests, the new version must be incorporated into the stable project to proceed with the development of the application.

An artifact is a product (or more activities) within the context of development of software or a system. In the SDPSS, it is defined as self-explained document and is used to customize the information related to the software development. Six artifacts make up the SDPSS: one for the general control of the process and five related to the specific area of development.

- **General Development Control** – Known as "umbrella" artifact, it is specific for the general control of software development. In this document, the registration of the information regarding the development, projected modules, project architecture, and changes made in each artifact in charge of having control of the development process as a whole are recorded. This artifact is created only once during the whole development of a specific project.

- **Version Requirements and Scope** – Used and/or modified in the Planning Phase, which is the first stage of the SDPSS development cycle. Here, the developer must register what has been planned for that iteration. Each iteration of the life cycle must generate a Requirement Document and a Version Scope.

- **Detailing of the Use Cases** – Based on the information collected and documented in the General Development Control and Version Requirements and Scope, this artifact can be modified even in the Planning Phase. It includes the application of use case diagrams and their specifications. Each iteration must, ideally, produce a complete artifact of Detailing of the Use Cases.

- **Version Codification Planning** – Can be modified in the Planning Phase and must be consulted and, if it is necessary, modified in the Development Phase. It is particularly important for the success of the development, because it defines all the details regarding the development of the source code itself, including the classes' diagrams. Ideally, each iteration must yield a complete Version Codification Planning artifact.

- **Tests Planning** – The generated source code serves as a basis for the application of the tests, which will be documented in the Tests Planning artifact. In this artifact, the developer has the possibility to document the whole test process that has been run. As described in the SDPSS life cycle, unit tests, followed by integration tests, must be carried out to guarantee that the built version is working.

- **Failures and Successes Recording** – Used at the final phase of the life cycle of the SDPSS. After a complete iteration, the researcher could have faced difficulties and possible failures, as well as successes in its development. This artifact allows the developer to register the failures and successes obtained in the development of each version. A Failure and Successes Recording document must be generated for each iteration of the life cycle.

These artifacts constitute an understructure of the development, and the researcher/developer has the option of omitting the optional information in these documents, which makes the SDPSS a highly customizable process. However, some parts of the artifacts are mandatory, with the intention of guaranteeing minimum documentation needed for the control of the development process.

Continuing with the development of the SDPSS, the present study is mainly focused on the implementation of the seventh step of the Humphrey's Methodology, with the intention of completing and publishing the process. This step tries to assure the process "correctness"or, in other words, to verify if the SDPSS produces positive results during the conception of a scientific software of academic nature. Accordingly, first, the SDPSS process has been automated through the creation of the webapp *PESC-V.1.0*. This tool is intended to model, systematize, and enable centralization of the artifacts produced by the process during the development of a determined project. With the automation, understanding as well as adaptation and management

of the development process become easier, enabling, in a simple way, the analysis of the data generated with the execution of the SDPSS during the validation process.

Another important functionality implemented in the webapp is the item “Review and Suggestions". From this functionality, the user, while using the process, can make suggestions related to the tool and process, which will be a data source to be used and analyzed during the validation and improvement of the process.

3 PROCESS VALIDATION

For a software process to guarantee efficacy of the product, it must be validated with the intention of identifying possible errors during its modeling, because the errors could turn software development into a chaotic process. Validation can accomplish many objectives, including guaranteeing the consistency of the formal process, because the execution of the process must be compatible with the process model described. This, consequently, increases the reliability of the results of any analysis executed with the formal model. Process validation can also be used as a certification tool, detecting differences between the desired behavior and the real one. Finally, validation can reveal when a process needs to evolve to accommodate activities and requirements of a new project (COOK; WOLF, 1999). Note that only the validation of the process takes place here; the verification is not an issue in the present study because there is no requirement or specification to be checked against in such a situation.

However, validation of a development process is not a simple task. One of the problems is that the validation is conceptually complex, because it refers to numerous questions, mostly subjective ones. The few researches carried out in this area are complex and dependent on the context in which they were applied. Moreover, no issued documentation proving or demonstrating the applicability of the validation techniques in a relevant quantity of real environments was found. Cook and Wolf (1999) as well as Moor and Delugach (2005) adopted the comparison between model and practice as the principle for processes validation, but in different ways. Moor and Delugach (2005) adopted a formalism based on the Graphs Theory, where model and practice are represented and compared.

On the other hand, the method adopted by Cook and Wolf (1999) was designed to measure the differences between the model and practice, known as the String Edition, where the number of insertions, exclusions, and symbolic substitutions (*tokens*) needed to transform a string (sequence of events of the process/practice) in another are analyzed. The ideas above are capable of producing interesting results to validate a process.

However, the use of the concepts of the Graphs Theory or String Edition can make the process validation of an agile development impracticable, because they advocate flexibility and easiness of use. Hsueh et al. (2008) utilized the concept based on simulation, and suggested pro-

cess validation through a game of projects management. The game is represented by a scenery of a software plant containing schedule and resources related to the project under development, and has a determined process associated to the development. In this scenery, it is possible to make the simulation and verify how the process treats a particular situation (NAVARRO; HOEK, 2005).

Sun, Du and Chen (2013) presented an approach to automatically discover explicit rules for software process evaluation from the evaluation histories. Each rule is a conjunction of a subset of attributes in a process execution, characterizing the reason why the execution is normal or anomalous. Sun, Du and Chen (2013) formulated this problem as a contrasting item set mining task and employ the branch-and-bound technique to speed up mining by pruning search space.

Ferreira and Moita (2010) also suggested an approach to compare the formal model and the execution of the process, but they adopted concepts based on software inspection. The main motivation of their method is to simplify and provide flexibility to the comparison of the formal model and its practice. Thus, this approach should be: (i) Generic; (ii) Simple; and (iii) Extensible. Through the application of the technique to a case study (FERREIRA; MOITA, 2010), evidences to these three characteristics could be demonstrated. In addition, this technique confirms that it can be applied in different processes with a low cost for the involved personnel with regard to any specific knowledge and bureaucracy in its use.

Furthermore, with respect to the validation approaches of the process described earlier, it can be clearly seen that there is no "best approach" for all of the processes to be considered, because the choice of an approach to validate a process will depend on the environment in which it will be implemented and on the experience of the people involved in the implementation of this process. As a result, it is desirable to analyze the approaches by having in mind the actual process and its applicable environment.

Regarding the SDPSS, once it becomes a development process designed to be used with scientific software, it has to comply with the principle of simplicity and has to be made to small teams with no need for considerable human or financial resources. If the process validation technique does not follow this intention, it can block its use. From the work of Gomes, Moita and Moreira (2011), it could be observed that the procedure that can be better applied to the SDPSS validation is the proposal by Ferreira and Moita (2010). The description of this technique is presented in the following section.

3.1 Inspection Technique applied for Development Process Validation

In the methodology developed by Ferreira and Moita (2010), the software development process inspection validation technique (VProcInsp) is performed through the following tasks:

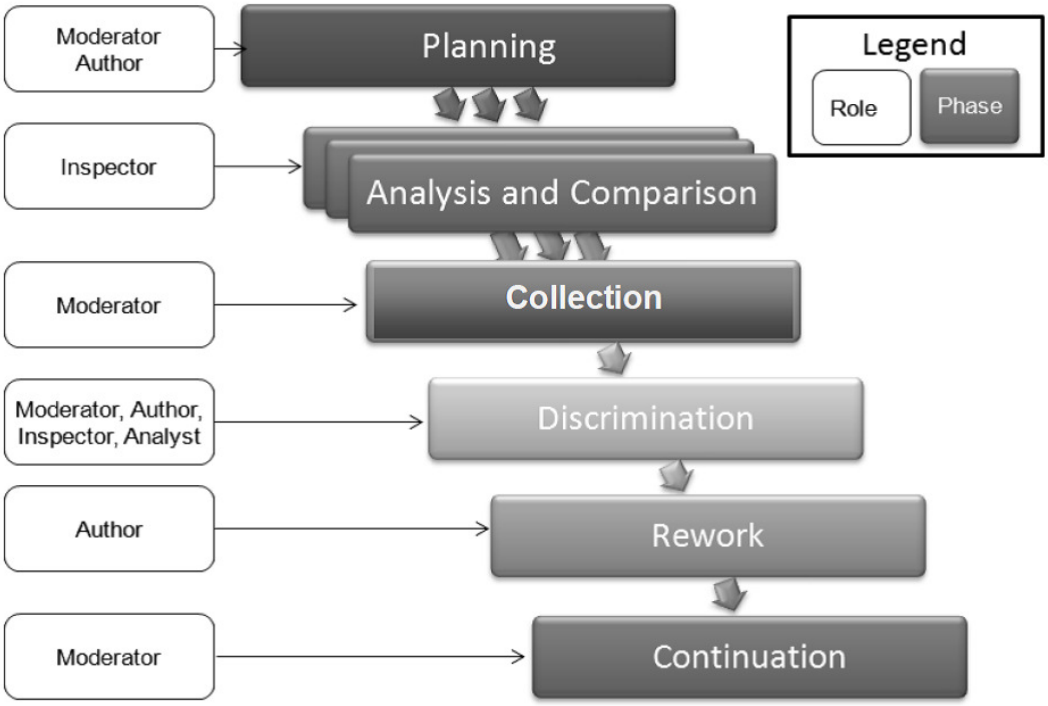
(1) Determine the software artifact that must be evaluated; (2) Identify the profiles and roles of the people who must be involved in this inspection; (3) Define the evaluation technique used to identify the faults in the artifact to be evaluated; and (4) Specify the process compound by the activities to be executed. The application of the Vprocinsp should be made by people who have experience in software engineering and the technique guarantees its reduced subjectivity due to the predefined steps to be followed during the validation of the process.

The description of the tasks needed for the application of the technique is as follows:

- *Determination of the Artifacts*: In the VProcInsp, the following artifact can be considered: documentation, electronic communication, version control software, compilers, and other software information (FERREIRA; MOITA, 2010).
- *Definition of the Role and Profile of the Validation Participant Team*: The VProcInsp has four roles with well-defined characteristics of acting: (1) Moderator – Independent person, individual, and impartial who coordinates the planning process, preparation, inspection behavior, and meetings; (2) **Author** – A person or a team in charge of the formal model of the software development process. The responsibility of the author is to explain and introduce process documentation; (3) **Inspector** – Responsible for identification, analysis, and comparison of the formal model of the process, with the process executed in practice. The inspector does not find errors in the artifacts, but analyzes whether the task proposed by the process is being carried out. (4) **Analyst** – A person or a group of people involved in the software development process, who use the process under validation.
- *Use of the Inspection Technique*: The main purpose of this technique is to validate the artifacts using a simple and methodic way and verify if the process is achieving what was proposed. According to Ferreira and Moita (2010), the most appropriated technique is the Checklist. However, they highlighted that the idea is not to create a fixed checklist, but to show a template with directives to allow the moderator to create the checklist in accordance with the needs of each development process. The inspection is used to compare the products and flow of the process actually executed with the products and the flow of the formal model.

Figure 3 shows the model proposed by the VProcInsp, followed by the description of this model.

Figure 3 – Inspection Model proposed by the VProcInsp



Source: FERREIRA AND MOITA, 2010

In the **Planning** phase, a professional who will exercise the role of inspection moderator is identified. The Planning is similar to Projects Management because in this activity, the team, terms, costs, and other functions concerning the management are defined. The main activity of the Planning phase is the configuration of the checklist to validation inspection. To configure the checklist, the Moderator should use the documentation or formal model of the process under validation with the support of the process author.

In the **Analysis and Comparison** phase, the inspectors individually analyze the artifacts and compare them with the formal model of the process. They are guided by the checklist, identifying discrepancy between the two sets of documents. It is important to highlight that most of the times the inspectors will have to investigate all the information produced during the development process. During the execution of the phase, the inspector must register and classify the discrepancies in a report.

In the **Collection** phase, the Moderator has access to all the lists of discrepancies through the Discrepancies Reports produced by the Inspectors in the phase of Analysis and Comparison. The Moderation can select discrepancies of these lists and discharge or classify them as replication if there is more than one discrepancy representing the same damage. When the discrepancy is discharged, it is withdrawn from the subsequent activities of the VProcInsp.

The Moderator, Author, and Inspectors participate in the **Discrimination** phase. In this phase, the discrepancies are treated as discussion topics. Each participant can add his or her comments related to each of the discrepancies, which remain available as a discussion topic, and

the Moderator and Author do not decide if the item really represents a discrepancy or not. It is important to stress that the solution for the discrepancies is not discussed in the Discrimination phase, and it is a task of the Author of the process in the **Reworking** phase, where the Author corrects the discrepancies detected during the inspection.

Finally, the Continuation phase consists of reviewing the material corrected by the Authors by the Moderator, who makes an analysis of inspection as a whole and revalues the quality of the inspected item. The Moderator has the autonomy to decide if a new inspection must occur or not.

4 THE SDPSS VALIDATION

The SDPSS validation questions if the initial process grants effectiveness to a scientific software development process of academic nature. With the purpose of choosing researchers to use the SDPSS in the creation of their software, in the present study, the process was presented to investigators in the *Intelligent Systems Laboratory* of CEFET-MG and to some researchers from other Brazilian universities. Five researchers placed themselves at full disposal to use the SDPSS as the development tool for the construction of their software of academic nature.

The initial idea was to select studies in which the software to be developed had different profiles, such that they could exercise, in diverse ways, the SDPSS process and artifacts. With this in mind, the following projects were selected: two M.Sc. developments – from the Computing and Mathematics Modeling Program of CEFET-MG and from the Computer Science Program of UFMG, respectively; one study from the Positivo University; and two undergraduate projects from the Computer Science Course from Faculdades Integradas de Caratinga, and from Automation and Control Engineering from UFMG. A brief summary of each project is given in the following paragraph to show the diversity of the software developed. It is important to mention that the students were from different research areas with completely different purposes and backgrounds.

- An Investor Psychology Study through an Automaton Cell (Investor's Psychology): This work points to a new field that opposes the efficient market hypothesis called behavioral finances. It shows, through empirical researches, that the investor's behavior often escapes from rationality. This hypothesis is studied using an automaton cell model with the purpose of analyzing how and how much an investment psychology, from investors, can influence stock market stability.
- Vehicles Routing Problem with Cross-Docking (Vehicles Routing): The problem with Vehicle Routing Problem with Cross-Docking (VRPCD) integrates characteristics from Cross-Docking strategy with the classical problem of Vehicles Routing Problem. VRPCD considers a scenery where a group of customers and suppliers (nodes), geographically

distributed with their respective demand of delivery and collection, must be attended by a team of vehicles that depart from a distribution center, called Cross-Dock. For this, a software was developed, whose objective was to determine the minimum number of vehicles and the best route for each one to visit all the nodes just once, thus minimizing the transport cost.

- Digital images transfer using J2ME and Bluetooth in mobile devices (Images Transfer): This work proposes the implementation of an application in JME using JSR of the Bluetooth, which allows the users to send their images directly to other cell phones that support Bluetooth Technology, in a fast way and without the need to use a computer.

- The Use of Artificial Neural Networks for Physical Conditioning Verification of a Professional Soccer Player (Physical Conditioning): This work uses an artificial neural network (ANN) to verify the physical conditioning of a professional soccer player, avoiding muscular exhaustion. The ANN training was made with the data of the players of Cruzeiro Esporte Clube (a well-known Brazilian soccer team), so that it could be used to provide real performance of the player in that circumstances.

- Development of a Tool for a System for Diagnostic of Alarms (Alarms Diagnostic System): This work presents the development of a tool to help alarms identification when they are badly configured as well as for the diagnostic of alarms systems from its log files. In the developed software, the alarm identification occurs in two ways: by statistical analysis recommended by EEMUA (Engineering Equipment and Materials User Association) 191, the norm that represents the main practices for alarm systems projects, or by using data mining techniques that identify rules of association among alarms. It must be emphasized that the result of data mining is presented in the form of a complex network, with which it is possible to identify alarms redundancy in an intuitive way.

Another point to be considered regarding the validation of the process was the profile of the researchers related to software development. Table 1 shows a short description of the knowledge in the area of each researcher, which can be relevant during the use of the SDPSS process. This knowledge can be classified as follows: knows, knows partially, and does not know, and were determined from interviews with the people involved in developments using the following topics as source: Object Orientation, UML, Software Process, and knowledge about the SDPSS. The profile is relevant because it can explicit the performance of the researcher related to the used process, and researchers with greater knowledge related to Software Engineering will probably lessen the effort while using some software development process.

Table 1 – Researchers' Profiles

Researchers	Object Orientation	UML	Software Process	SDPSS Knowledge
P1	Knows Partially	Does not know	Does not know	Knows Partially
P2	Knows	Knows	Knows	Knows Partially
P3	Knows	Knows	Knows	Knows
P4	Knows	Knows Partially	Does not know	Knows Partially
P5	Knows Partially	Knows	Knows Partially	Knows Partially

The software that supports each of the projects mentioned was developed using the SDPSS to generate results that can be analyzed by means of the VprocInsp to investigate whether the process can ensure its correctness and efficiency. It is noteworthy that in all the test cases, the webapp PESC-V.1.0, described in Section 2.1, was used.

4.1 Description of the Validation of the SDPSS

As the main idea was to validate the SDPSS with the narrowest possible margin of subjectivity, all the necessary tasks for the construction of the technique were performed, even with some restrictions or safeguards. In the following, each task and the considerations made are listed.

- **Determine the process artifacts**

The SDPSS is composed of six artifacts, five produced at each iteration in the development and only one for the general control of development as follows: General Development Control, Requirements and Scope Version, Detailing of the Use Cases, Plan Version Coding, Test Planning, and Successes and Failures Recording.

- **Definition of the role and profile of each validation team participants**

At this point, the participants of the validation process were defined. The first author of the present paper had the role of Moderator and Author. This decision was taken due to his knowledge regarding the SDPSS as well as the technical validation processes, mainly the VprocInsp. It was decided that two inspectors would be adequate for the application of the technique. Inspector 1 was responsible for the planning phase and Inspector 2 was responsible for the stages of development testing and deployment, totaling three artifacts for each inspector. Importantly, the validation team was the same for the five case studies, with the exception of the analyst, who was represented by the researcher or the developer of each case study. The inspectors were selected based on the experience with the use of formal software development processes as well as with aspects related to Software Engineering.

- **Taxonomies of the faults**

VprocInsp used the following taxonomy of errors: omission, ambiguity, and inconsistence. Any error that cannot be classified within these categories was classified as "other".

- **Technical inspection**

The technical inspection used to validate the artifacts of the SDPSS was the Checklist, whose main objective was to validate the artifacts in a methodical and simple manner, and verify whether the process reaches its expectation. Five checklists were set, one for each artifact of the SDPSS. These checklists were created by the Moderator/Author based on Software Engineering concepts and according to each artifact contained in the formal process of the SDPSS.

- ***Modus operandi***

During the inspection, the "real life" roles and process flows could be examined and compared with the roles and flows of the formal process. VprocInsp was exercised, which considered the steps presented in Figure 3: Planning, Analysis and Comparison, Collection, Discrimination, and Rework.

Finally, after the application of the validation process for the SDPSS, some inconsistencies were found in the formal process. These inconsistencies are described in the next section.

5 DISCUSSION OF THE RESULTS

Table 2 shows a summary of the discrepancies found during the execution of the process validation technique VProcInsp, separated by artifacts of the SDPSS. It is important to mention that the technique validated the items that appeared in the SDPSS formal process. However, if the project needed something else for the development of the software, which did not appear in the process, then the lack of this item was not taken into consideration in this table. Thus, the “Evaluation and Suggestions” option of the SDPSS webapp tool allows the researcher to relate any extra need for software development (including any missing items that might have been detected). With this possibility, the process validation team could verify a possible requirement of the researcher, which was not observed by the process.

Table 2 – Summary of the discrepancies found during the SDPSS validation

Artifact / Project	Investor Psychology	Vehicle Routing	Image Transfer	Physical Conditioning	Diagnostic Alarm Systems
General Development Control	Item Description Details of Architecture.	Item Description Details of Architecture			Item Description Details of Architecture
Requirements and Scope Version	Item Description Requirements and Scope Version Ambiguity: "Scope Release" and Expected Features	Item Description Requirement sand Scope Version Ambiguity: "Scope Release" and Expected Features	Item Description Requirements and Scope Version Description of the item "Special Requirements" is not clear. Ambiguity: "Scope Release"and Expected Features	Item Description Requirements and Scope Version Ambiguity: "Scope Release" and Expected Features (should be optional)	Item Description Scope Version Item Description viability Item Description Requirements and Scope Version
Use Case Detailing	Description of "extension points"	Description of "extension points"	Description of "extension points"	Description of "extension points"	Description of "extension points"
Plan Version Coding	No information was found for all items of the classes details	No information was found for all items of the classes details	No information was found for all items of the classes details	No information was found for all items of the classes details	No information was found for all items of the classes details
Test Planning	There were no documented interface test, source code, performance, and integration.	There were no documented interface test, source code, performance, and integration.	There were no documented interface test, source code, performance, and integration.	There were no documented interface test and integration.	There were no documented interface test, source code, and integration performance
Successes and Failures Recording	-	-	-	-	-

The validation of the process took place through the artifacts generated during the software development; in other words, all the activities described in the five artifacts that compose the SDPSS were validated.

The process of validation began from the General Development Control artifact, in which it was detected that the three researchers did not understand how the process treated the item Software Architecture. To solve this point, it was suggested that the description of this item should be rewritten to include examples to eliminate ambiguities.

With regard to the artifact Version Requirements and Scope, it was pointed out, in all the developed projects, that the items Version Scope and Expected Functionalities were causing ambiguity, as well as complicating the understanding of the researchers. It was also cited by all the researchers that the item Requirements and Version Scope was not described in a clear and concise manner. The developer of the project "Alarms Diagnostic System" indicated that he did not understand the description in the item Special Requirements, which did not allow its production. The item Viability was not filled by the researcher of the "Vehicles Routing,"who claimed that he did not understand its description. From these analyses, it is suggested that the

items previously cited be restructured in the Version Requirements and Scope.

With regard to the artifact Detailing of the Use Cases, the item Extension Points was not understood by any of the researchers and could not be used. Another item that was not comprehended by those responsible for the "Alarms Diagnostic System" was the Special Requirements. Hence, it is recommended that these two items be revisited in such a manner to facilitate its understanding by the researchers.

In the artifact Version Codification Planning, all the developers claimed that during the filling of the item Class Detailing, topic Attributes, Operations, and Responsibilities should not be compulsory, because some classes might not contemplate all these items.

The artifact Tests Plan was the most questioned during validation by the researchers. It was pointed out that none of the projects contemplate, in a documented form, the items Source-Code Testing, Integration Testing, Interface Testing, and Performance Testing. It is important to highlight that the items related to source-code, interface, and integration tests are set as mandatory by the SDPSS, which may be considered as an inconsistency with the simplicity and flexibility advocated by the process. In general, the researchers related the lack of information about the techniques to be used by the tests indicated in the artifact Tests Planning, making its production difficult.

Finally, in the validation regarding the artifact Failures and Successes Recording, no inconsistencies were found, indicating that all the items of this artifact are relevant in the development process.

At this point, validation of the SDPSS formal process could be regarded as completed; in other words, all the items of the process were checked and tested during the development of the case studies.

With the purpose of aggregating more quality to process validation, the item Evaluation and Suggestions of the SDPSS webapp was also analyzed to examine the expectations of the researcher during the development of his or her software.

Some researchers related that the process does not mention anything with respect to the modeling and diagrams of the database, which is extremely important for software development, mainly considering any possible change in the software. These researchers also suggested including an option to insert more UML diagrams in the process, because some of them had to use extra diagrams in their developments.

Another point raised was regarding the artifact Tests Planning. The researchers reinforced that similar to the fact that each one should know his or her own research work, the tests could be created according to their particular needs, so that the existing compulsory status could be set as optional, and still, in this way, work for the description of the techniques to be used to support the researcher in their use.

Lastly, it should be stated that before releasing the SDPSS process for the general ap-

plication (and evaluation) by the scientific society, all these adjustments must be made in the formal document of the process as well as in the tool that implements it. It is important to highlight that the validation technique used obtained good results and was very simple to be applied during the process of validation of the SDPSS artifacts.

6 CONCLUSION

In the present study, the SDPSS was validated. The validation was carried out using the validation inspection technique VProcInsp, and was applied to five projects that develop scientific software. Once the selected projects were very distinct and related to different fields of applications, it was possible to widely "exercise" the process. Another interesting observation was the diverse formation of each researcher involved in this practice, because the difficulty of some of them with respect to Software Engineering terms was observed; they should appeal to the literature and, thus, need some added effort for the development of the software. This fact will be further exploited in a future paper. However, even with such a problem, the SDPSS could be used, which confirms its simplicity.

Although the SDPSS has shown some inconsistencies after the validation process, it is important to underline that despite the adjustments presented in Section 5, the process offers strong evidences that it can contribute to the academic society in the development of scientific software. It is also important to draw attention to the improvement and evolution of the SDPSS that can be obtained from the feedbacks from people who use it to provide increasing consistency and credibility to the process.

Finally, it must be stated that the case studies used during the validation process are not adequate to assure that the process is fully validated. However, they can be sufficient to give evidences of the characteristics that enable its use by the scientific community.

References

COOK, Jonathan E; WOLF, Alexander L. Software process validation: quantitatively measuring the correspondence of a process to a model. **ACM Transactions on Software Engineering and Methodology**, ACM, New York, v. 8, n. 2, p. 147–176, 1999.

FERREIRA, Bruno; MOITA, Gray F. Inspection technique for the validation of software development processes. In: IOP PUBLISHING. **IOP Conference Series: Materials Science and Engineering**. [S.l.], 2010. v. 10, n. 1, p. 1–9.

GOMES, J. O.; MOITA, G. F.; MOREIRA, A. B. Comparative analysis of different approaches for the validation of software processes. In: CILAMCE IBERIAN LATIN-AMERICAN CONGRESS ON COMPUTATIONAL METHODS IN ENGINEERING, 32., 2011, Ouro Preto, Brazil. **Proceedings...** Ouro Preto: UFOP, 2011.

HSUEH, Nien-Lin et al. Applying uml and software simulation for process definition, verification, and validation. **Information and Software Technology**, v. 50, n. 9-10, p. 897–911, 2008.

HUMPHREY, Watts S. **A discipline for software engineering**. 1. ed. [S.l.]: SEI series in software engineering, Addison-Wesley, 1995.

KALUS, Georg; KUHRMANN, Marco. Criteria for software process tailoring: A systematic review. In: ICSSP INTERNATIONAL CONFERENCE ON SOFTWARE AND SYSTEM PROCESS, 8., 2013, San Francisco, CA, USA. **Proceedings...** San Francisco, CA, USA, 2013. p. 171–180.

MOOR, A. D.; DELUGACH, H. Software process validation: Comparing process and practice models. In: ICCS INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE, 5., 2005, Atlanta, USA. **Proceedings...** Atlanta, USA, Emory University, 2005. p. 533–540.

NAVARRO, Emily Oh; HOEK, André Van Der. Software process modeling for an educational software engineering simulation game. **Software Process: Improvement and Practice**, Wiley Online Library, v. 10, n. 3, p. 311–325, 2005.

PEREIRA, J. M.; MOITA, G. F. The conception of a specific development process for scientific software. In: SIMPóSIO DE MECâNICA COMPUTACIONAL, 8., 2008, Belo Horizonte. **Proceedings...** Belo Horizonte, PUC Minas, 2008.

PRESSMAN, Roger S. **Software engineering: a practitioner's approach**. 7. ed. [S.l.]: Science Engineering & Math, 2001.

PURRI, M. M. S. **Initial study and proposals for the definition of a specific development process for scientific software**. 2009. Master's thesis — Centro Federal de Educação Tecnológica de Minas Gerais - CEFET-MG, Brazil.

SILVA, D. E.; SOUZA, I. T.; CAMARGO, T. Methodologies agile software development: Application and use of scrum methodology in contrast to the traditional model of project management. **Magazine Applied Computing**, v. 2, p. 39–46, 2013.

SUN, C.; DU, J.; CHEN, N. Mining explicit rules for software process evaluation. In: ICSSP INTERNATIONAL CONFERENCE ON SOFTWARE AND SYSTEM PROCESS, 8., 2013, San Francisco, CA, USA. **Proceedings...** San Francisco, CA, USA, 2013. p. 118–125.