SSP: A Simple Software Process for Small-Size Software Development Projects

Sergio F. Ochoa[1], José A. Pino[1], Luis A. Guerrero[1], César A. Collazos[2]

[1] Department of Computer Science
Universidad de Chile
Blanco Encalada 2120, Santiago, Chile
{sochoa, jpino, luguerre}@dcc.uchile.cl

[2] FIET, System Department
University of Cauca
Popayán, Sector Tulcán, Colombia
ccollazo@unicauca.edu.co

Abstract. A large number of software development projects in Latin-American countries are small-size, poorly defined and time pressured. These projects usually involve under qualified people. Provided that well-known software development models have shown limited applicability in such scenario, developers usually carry out ad-hoc software processes. Therefore, the obtained results are unpredictable. This article presents a Simple Software Process (SSP) for small-size software projects involving under qualified people. The proposal is motivated by current practice in Chile. SSP proposes a step-by-step process which structures the development activities and it improves the process visibility for clients and team members. Furthermore, SSP formally includes "the user/client" as an active role to be played during the project. This process has been used in 22 software projects and the results are encouraging.

1. Introduction

Most software development projects in Chile are information systems of small or medium size (1-2 months or 3-6 months) [17]. Typically, these projects involve time pressured activities and clients reacting just when they detect the need for a software solution [9, 21]. For that reason, these projects have a high rate of volatile requirements [17].

Typically, qualified developers are involved in large or medium-size projects whereas small software projects are carried out by under-qualified or inexperienced

software developers [17, 21]. The reactionary development scenario and the lack of clear guidelines to face the process, push developers to follow an ad-hoc development process. A recent study carried out by Sacre concludes that software processes in Chile tend to be chaotic and unpredictable, because they do not have a guiding development model [17]. Besides, each development is influenced by variables like type of project, client and development team. It shows how immature are the processes in this scenario. As a consequence, software projects in this scenario cannot assure the development time and cost nor the quality of the final product [14, 17].

The heavyweight software methodologies are limited to support such scenario. This is because they involve several stages and roles that require an important amount of communication and coordination in order to get a final product. This required bureaucracy jeopardizes the applicability of such software models.

On the other hand, there are the lightweight or agile methodologies that could have an interesting applicability to the described scenario. However, the high clients/users availability required to support the development process makes these processes unsuitable. The need to develop in an asynchronous and distributed way is another important limitation to adopt lightweight software processes in this scenario.

Alternatively, in order to solve the stated problem, this paper presents a software process called Simple Software Process (SSP), which has been designed to guide small-size software development projects in immature scenarios. This methodology was slowly evolving, as experience with real word cases was accruing.

Next section presents the critical issues which give rise to most problems in the software projects. Section 3 presents the related work and analyzes the applicability of the best known software processes to immature scenarios. Section 4 presents the proposed software process. Section 5 analyzes the results obtained after applying the methodology. Finally, section 6 states the conclusions of this work and the future related activities.

2. Critical Issues

Based on the studies of the local software industry conducted by Sacre [17], Stein [21] and IDC [10] during 2002, and based on several reported experiences and authors' experiences, a set of critical issues has been identified. Some of these issues present facts, which may not be changed and thus, any proposed solution must cope with them. These issues are the following ones:

Deadlines Determined by Need. Typically, the project deadlines are determined by the need of the client for deploying the software solution in his/her organization. Typically the available time is shorter or equal than the required one to do a good job. Therefore, it should be assumed deadlines will be difficult to reach and work will be done under pressure.

Asynchronous and Distributed Work. Most team members usually work in a distributed and asynchronous setting with little time dedicated to the project. Each member has allocated a time quantum to carry out the work and he/she has little time for coordination and integration activities [21].

Under Qualified Developers. Typically, development teams are composed of a senior engineer acting as project manager, and a set of senior students of Computer Science and Technology or junior engineers with little expertise in software development and teamwork [17, 10]. Although technically they are able to tackle a problem, they have difficulties to work as team members, to interact with users/clients and to identify/manage requirement changes and to handle risk and unpredictable problems. Most of them have little time assigned to the project and they work in an asynchronous and distributed way. Each member of the team plays more than one role, e.g., project manager/tester or analyst/programmer, but the rights and duties associated to each role are not explicitly defined [21]. In such cases, the project manager assigns activities based on his own best judgment [9]. As an example, we can mention that it is possible to observe programmers making design decisions, or that some tasks are simply not carried out because the project manager forgot to assign them.

On the other hand, there are further critical issues that need to be managed. The inadequate management of these issues produces most problems appearing in this type of projects. The critical issues to be managed during small development projects in immature scenarios are the following ones:

Clients Availability. Usually, the client has little time to interact with the development team. Many tasks involving the client, such as information providing, decision-making, review of prototypes, are accomplished late because his/her lack of time [21].

Requirement Stability. Most of these software projects are consequences of clients' reaction triggered by the identification of a need for a software solution. Therefore, software projects are not well conceived and matured at start time with the requirements elicitation. It produces permanent changes of the requirements and a lack of visibility of the software project [17]. Clients feel they have the right to adjust the requirement without paying extra money because the developers are not realizing what they want. This is one of the main causes of conflicts between clients and developers.

Coordination Activities. Most team members are not full time dedicated to the project and they work in an asynchronous and distributed way. There are no clearly defined roles in the team and there are no clear rights and duties associated to each team member [21]. Project managers assign activities to team members based on their own best judgment [9]. In addition, they are in charge of coordinating these team members activities based on some ad-hoc strategy. In a development scenario with many low dedicated, distributed and beginner developers is too easy to lose control of the project.

Project Visibility. Typically there is not enough time to do a proper development, therefore management and control activities are superficially done. There is high unplanned parallelism related to the tasks of team members, which originates an unnecessary workload and conflicts of scope. Typically, it is difficult to determine the advance status of the project and the workload required to finish it. It generates conflicts with clients and within the development team [17, 9].

Effort Estimation. The project duration and the initial functionality are quite fixed; therefore the effort estimation is reduced to a money issue [14]. The time available to develop the project is directly related to the client's urgency to have the products. Sometimes such duration is not viable to get good products, but ignorance about the productivity of these work teams drives the company to take high-risk projects. Generally, these projects are finished late and/or the resulting products do not satisfy the client expectations [14].

Product delivery. It is often clear what the final product the project should deliver in terms of software code, but the same cannot be said for the corresponding documentation. The process is strongly focused on delivering software, but the documentation is either incomplete or totally forgotten. In addition, the contents of intermediate products (specification of requirements or document design) are not carefully studied to verify if they are appropriate for the applied development style. On the other hand, there are many development teams that produce only the requirements specification and the software of the final product. Generally, this lack of formal intermediate products is the main cause for communication and coordination problems within the work team [17]. Moreover, the informality to elicit, specify and use requirements in these projects is the most important cause for a conflictive climate both among developers themselves and between them and the client [21].

Although these critical issues are the source for several problems in this type of projects, there is a lack of guidelines to manage them. In addition, well-known web development models seem to be problematic to be used in contexts such as the Chilean one, because of they are so heavy weight as to be carried out in short time using novice developers. This situation forces software developers to use handmade procedures to develop the Web software products. Next section presents the related works and the strengths and weaknesses of several proposals that could be used to solve the problem.

3. Related Work

Most models reported in the literature for software development are oriented towards mature development scenarios. Some of them are known as heavyweight software models because of the bureaucracy required during their application to a project. On the other hand, there are lightweight or agile software processes that involve minimal bureaucracy, but high interaction among team members and between team members and the users/clients.

Heavyweight software processes seem to be problematic to be used in immature contexts, because they are so difficult to be carried out in a short time period by using developers. Some of the most representative heavyweight software processes include the following ones.

OOHDM (Object-Oriented Hypermedia Design Methodology). This is a development methodology for hypermedia applications, including Web applications [18, 20]. OOHDM offers a clearly defined process, which can be

adopted using an incremental model or a prototype based model. It proposes five basic steps to carry out the development: *requisite elicitation, conceptual design*, *navigational design, abstract interface design* and *implementation*. Although the process is clear, it is not easy enough to be used by beginners in short time periods.

WSDM (Web Sites Design Method). This methodology uses a user-centered design strategy [4]. The design is driven by the views of different user-classes instead of being data-driven. This method is limited to "Kiosk Web Sites", i.e., Web applications that only display data, and can be navigated through themselves. WSDM is a variant of the waterfall model that involves four phases: *user modeling*, *conceptual design*, *implementation design* and *implementation*. One of the most important advantages of this model is the application of user requirements as a guide for the development process. As regards disadvantages, since the model is based on the waterfall model, WSDM is affected by its typical problems [2]. In addition, it is only focused on the design process and does not give a clear support for roles, asynchronous and distributed work and coordination activities.

WebComposition. This model describes a consistent approximation to the Web Applications development based on components [8]. Basically, this model follows a spiral process that involves three phases: *analysis and planning*, *design and implementation.* The process is simple and provides feedback about process and product in a continuous way. One limitation of this model is the disregard for user requirements as a guide for the development process. It is focused on the design of the product, and it forces to use WCML (WebComposition Markup Language) [7] to represent that design. In addition, this model supports a rapid development only if the work team has an available library of reusable components, which are appropriate to build the new product.

WebE (Web Engineering). WebE is a general model described by Pressman [16], which follows an evolving approach including six stages: *formulation, planning, analysis, engineering, Web Pages development, testing and user evaluation.* Although this model is well conceived, it is heavyweight to be carried out in a short time. Besides, it has some restrictions such as: it does not perform requirements management; it demands a great effort for product design; each phase requires specialists; and the roles of the work team are not clearly defined.

RUP (Rational Unified Process). RUP provides a disciplined approach to assigning tasks and responsibilities within a development organization [12]. Its goal is to ensure the production of high quality software that meets the needs of its end users within a predictable schedule and budget. This software process involves the following phases: *inception*, *elaboration*, *construction* and *transition*. Team members work toward the milestones that mark each phase completion by performing activities organized into nine disciplines.

These models were not designed to support small software projects carried out in immature scenarios. For that reason their complexity, formalism, lack of support for quick developments or lack of a formal participation of the client during the development process restrict their applicability. However, there are lightweight software processes or agile methods that can be used to overcome limitations of

traditional software processes [6]. Some of the most known agile methods are the following ones:

Extreme Programming (XP). XP was created for small and medium size software projects where requirements are vague, change rapidly or are very critical [1]. XP was designed having in mind the problems with traditional programming methodologies with respect to deadlines and client satisfaction.

Scrum. Scrum was not conceived as an independent method, but a complement of other agile methods [19]. Scrum stresses management values and practices, and it does not include practices for the technical parts (requirements, design, implementation). For that reason, Scrum can be used in conjunction with another agile method. Scrum is a management and control process that implements process control techniques.

Crystal. Crystal is a family of methodologies created by Cockburn [3]. They are based on the fact that, comparing software construction with an engineering process makes us think about software "specifications" and "models", about its completeness, correctness and operation. The most exhaustively documented Crystal methodology is Crystal Clear (CC). CC can be used in small projects with medium criticality, although it can also be applied to critical projects if it is properly extended.

Feature Driven Development (FDD). FDD is an agile, iterative and adaptive method that it does not cover the complete software life cycle, but only the design and implementation phases. It is considered adequate for major mission critical projects [15]. FDD applies an iterative development with the best found practices to be effective within industry parameters. It stresses quality aspects and it includes small tangible deliverables, together with the precise control of the project progress.

These agile software methodologies do not use strict phases but they include a series of recommendations which aim at easing up the development [1, 11]. In addition, they substitute the strict documentation for an intense level of communication among clients and developers. However, the lack of time from clients and the problems of communication and coordination noted in previous projects jeopardize these development approaches. Besides, they have been proven inappropriate for developments in which the team members work in an asynchronous and distributed way. The following section presents the SSP methodology, which has been specifically designed to support small-size software development projects in the above mentioned scenario.

4. The Simple Software Process

The Simple Software Process (SSP) proposed in this paper intends to be appropriate enough to support the development of small information systems, in immature scenarios. The first version of this process was defined in 1998 to support software

development projects carried out by computer science undergraduate students at the Pontificia Universidad Católica de Chile.

The course where this experience took place was ICC2152 - Software Engineering Laboratory (10th term). In this course, students are grouped in teams and one role is assigned to each member. Responsibilities and rights of each role are specified in SSP. During 16 weeks the teams develop a real software application and interact with real clients and users. Over fifty small-size projects have been developed using SSP.

Since 2001, SSP has been also applied in the Universidad de Chile, in a course similar to the previously mentioned one. This work reports only the last twenty-two projects which have been carried out by undergraduate and graduate students from Universidad de Chile, in the course CC51A – Software Engineering (10th term), which keeps the same development scenario.

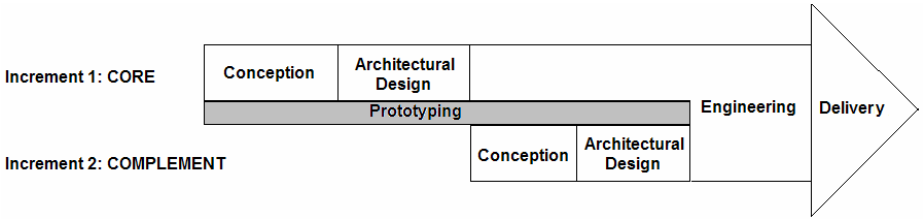

Fig. 1. Structure of SSP model

This development process involves two increments (Core and Complement) and each one is composed of four phases (Figure 1): *conception*, *architectural design*, *engineering* and *delivery*. The reason to propose two increments is because past experiences have shown that controlled two-steps approaches provide good results in short-time projects.

The first increment involves about 70% of user requirements and 100% of quality requirements. The second increment addresses the residual user requirements, which usually are not clear by the time the project starts. Thus, delays because unclear and changing requirements are reduced. The requirements of the second increment should be aligned with those defined in the first increment. Otherwise, a negotiation instance will be required.

Usually, the time spent in the development of the first and second increment is also around 70% and 30% respectively. The SSP approach involves a little work of integration, which has not relevant impact on the project schedule. In addition, the products to be integrated have been conceived and specified for fast integration. It allows developers to work asynchronously and in a distributed way avoiding delays caused by unclear requirements.

Furthermore, the prototyping during the development is a service that supports each phase in order to make it agile, improve the quality of the obtained products, and reduce the anxiety of clients. The next section describes the SSP phases and the dynamics of the development process. After it, section 4.2 presents a brief description of the roles involved in the development.

4.1. Phases of the model

In contrast to other development methodologies, the parallel work with low interactions among team members is fundamental in SSP. The restrictions on development time force team members to optimize the process, by maintaining low interaction among them, and working in an asynchronous/distributed way. In the following sub-section, the four phases of SSP are presented.

Conception. This stage has two goals: (1) to define the project viability, and —if it turns to be viable— (2) to specify the user requirements which will guide the development process. The project viability is identified through an effort estimation methodology called CWADEE (Chilean Web Application Development Effort Estimation) [14]. If the project turns viable, the collected information is used to design the elicitation process, which has two stages. The first stage is oriented to capturing the most important and stable user requirements. With this information, the developer may create prototypes that are used to verify, validate and redefine such requirements with the user-clients. The second stage is oriented to capture those requirements that are contradictory, conflictive or not clear enough. The prototype developed for the first stage is updated in order to support the prototype revisions with users and clients. Typically, these two stages are enough to identify the user requirements. Then, if needed, the development effort estimation could be adjusted. Finally, a User Requirement Document (URD) is created and validated through rapid prototypes. This document, like other ones proposed by SSP, is clearly specified and it is simple to write. Furthermore, during this phase, a set of test cases is built and documented in TCD.

Architectural Design. The inputs to this phase are the URD and the last prototype of the system. The phase main goal is to define the product structure in terms of subsystems, relationship among subsystems, information structure, system navigation and basic look-and-feel. It also specifies the operational environment of the system. This information is included in the Architectural Design Document (ADD) which is the result of this phase. During this stage programmers work in parallel with designers, by having these latter ones keep the coordination of activities and process control. Thus, when this phase ends, the obtained prototype is used to test the designed architecture with users and clients. Such prototype includes the look-and-feel, the navigation pattern and the raw functionality of the system.

Engineering. This phase uses the ADD to generate a detailed design that is implemented directly on the current prototype. The usability is the motivation for this phase, and the main goal is to get a product that is usable. During the development of the first increment, the programmers implement as much as possible in order to reduce the risks and to validate the usability of the Web application. During the second phase, the additional functionality is implemented, and both the complex functionality and the component integration are carried out. Eventually, some designers can participate in this phase as consultants in order to ensure the product usability. Upon finishing this phase, a usable product meeting the increment requirements should be obtained.

Delivery. The delivery phase is focused on installing the product in the user/client premises, to evaluate the acceptation level and to carry out minor adjustments if necessary. This phase is short and it is in charge of the programmers.

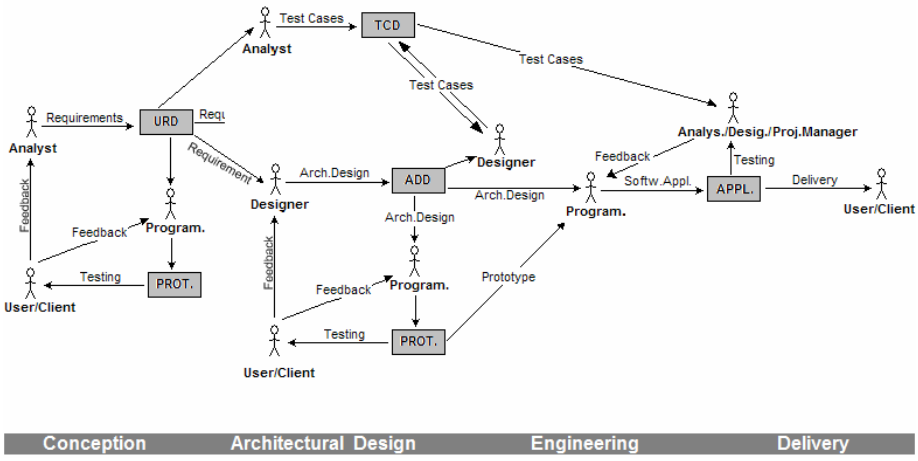

Fig. 2. Example of SSP dynamics

Figure 2 shows the dynamics of an SSP increment. The diagram shows three cycles in which it is possible to do testing and get feedback from the clients/users/team members. The first cycle is related to the conception phase, the next one is related to the architectural design phase and the last one is related to engineering. The process dynamics can be adjusted according to team member skills and roles that are present in the team work.

4.2. The roles

SSP demands for six roles to be assigned to group members: *project manager*, *analyst*, *designer*, *programmer*, *tester* and *user/client*. Although the user and client are not formally part of the work team, they play a key role during the development in order to help accomplishing the project scheduling. SSP formally proposes to include this role in the development process as a way to assure a quantum of user/client's time. Experiences using SPP indicates that it is a key factor to consider in order to have normal project execution. The formal participation of users/clients allows them to be conscious of their responsibilities. Moreover, the rest of team members are conscious of the user/client's role and the formal interactions required with these new members of the team. Provided that roles and interactions among the participants are well specified, a common understanding is created and maintained during the execution of the project. It increases the project visibility for all people involved in the development.

A team member could play a maximum of two roles during the project in order to avoid bottlenecks. However, just some roles combinations are recommended. For example, it is not recommended that a person may have the roles of programmer and tester, because testers have to review the programmers' work. Next, a brief description of the roles considered in SSP is presented.

Analyst. The analysts are responsible for the conception phase. They have to (1) establish if the project is viable or not, and (2) specify the user requirements in the URD (User Requirements Document). This document is a simplified version of the ESA Software Engineering Standard proposal [5]. The analyst-programmer or analyst-tester role combinations have shown to be appropriate if a person has to play more than one role.

Designer. The designers are in charge of the architectural design phase, which produces the ADD (Architectural Design Document). It includes the architectural design of the software application and operational environment. Moreover, it includes the design of the application look-and-feel and navigation. Designers also collaborate with the programmer during the engineering phase by testing and improving the product usability. Besides, they can adjust or add test cases to TCD (Test Cases Document). A person whose main role is designer can also play a programmer or tester role.

Programmer. The programmers are in charge of the engineering phase. They are responsible for the development of fast prototypes to be shown and the final product. Usually, they participate during the architectural design phase in order to assure that designs may be quickly implemented. A person whose main role is programmer can also play an analyst or designer role.

Tester. The tester is usually a distributed role, which is played by several members along the development process. For example, analysts can play a tester role when the conception phase has concluded. This role is responsible for specifying the test cases and for checking whether the products adhere to the specifications. Typically, the tester generates the TCD and reports the testing processes. The process presented in Figure 2 shows roles that can also act as tester during each phase of an increment.

Project Manager. The project manager plans, coordinates and controls the activities of the team members. The manager can also act as tester during part of the process and he/she typically acts as a communication interface with the client/user.

User/Client. The users and clients are in charge of (1) providing information and requirements of system to be developed, and (2) providing feedback to developers about the interim products that are delivered during the process. The software developers have internal check points with the users and clients every week in order to diagnose the project advance. Such meetings are formally scheduled and they take 10-20 minutes. Attendance to these meeting is part of the duties of users and clients.

Experiments performed in two Chilean universities are presented below. The experimentation scenario is similar to the one characterized in section 2.The obtained results are not conclusive enough; however they show the web development process in immature scenarios can be controlled in order to be predictable.

5. Experimental Results

Various versions of SSP have been used in more than 50 projects since 1998. This paper reports only the 22 last projects which show the results of the SSP current version. This software process has been used to support software development in software engineering courses taught at two major Chilean universities: Pontificia Universidad Católica de Chile and Universidad de Chile.

Typically, these courses are taken by advanced undergraduate and graduate students of computer science. As previously mentioned, students are grouped in teams of 4-6 people. Each team member had to play at least a role, by considering that all SSP roles must be covered. Then, a real project is assigned to each team. The projects involved participation of real clients and users. Although the scope and main requirements of the projects were previously agreed between instructors and clients, the team members had to negotiate the projects scope with their clients in order to make effective the developers estimations. Each team had 16 weeks to develop and deliver the final product.

At start time, the work teams had to define SSP adaptation to conduct the development process. The communication/coordination infrastructure supporting the team members included email, telephone, a CVS (Concurrent Versions System) and a document describing the roles and the interaction protocol. Students were free to use these or other coordination tools.

Instructors defined three main check points for each project execution: upon finishing the conception phase during first increment, upon ending the conception phase during the second increment, and upon completing the engineering phase during both increments (core and complement). In order to diagnose the projects advance, a formal technical review was conducted during each check point. The reviews took 60-90 minutes by project. These instances were used to get part of the results reported in table 1.

Results shown in Table 1 correspond to those obtained in software projects developed from first term 2003 to second term 2005. In order to present the results the projects were classified by the instructors according to size and complexity, based on the amount and complexity of user requirements. The following project categories were identified: Very Small size - Medium complexity (VSM), Very Small size – Complex (VSC), Small Size – Low complexity (SSL), Small Size – Medium complexity (SSM). For each project category, it is presented:

a) the number of initiated projects,

b) the number of projects under production -successfully finished- ,

c) the number of members per work team,

d) the average and standard deviation of the spent man-hours,

e) the average and standard deviation of clients'/users' assessments about the obtained product,

f) the average and standard deviation of team members' assessments about SSP as support for the development process,

g) the average and standard deviation of the experts' assessments about the quality of the final product,

h) the average and standard deviation of the team members' assessments about visibility of the project provided by SSP, and

i) the average and standard deviation of the clients/users' assessments about visibility of the project during the development process.

The two first assessments (items e and f) were carried out by using questionnaires designed with the method proposed by Zapata et al. [22], and the third assessments (item g) was done using an extension of the 8-issues questionnaire proposed by Nielsen [13]. The values range between 1 and 10, the higher the better.

Results show the SSP is predictable in terms of time, because most projects were completed and put into production. The man-hour values are stable enough according to project types as to support realistic estimations, regardless of the team work. The clients' and experts' assessments indicate that good quality products can be obtained. The work teams' opinions show a high level of satisfaction when using SSP to guide the development process. The same occurs with the project visibility as seen by clients/users and team members. The low clients/users commitment was the common factor in those projects that were not put into production.

Table 1. Experimental results

Item	Category/Issues	VSM	VSC	SSL	SSM
A	Number of Projects	4	6	7	5
B	Number of Completed Projects	3	5	7	4
C	People by Work Team	4	5 – 6	4 - 5	5 – 6
D	Man Hours / Standard Deviation	248 / 35	367 / 71	285 / 32	389 / 68
E	Clients-Users Assessment / Std. Deviation	8.5 / 0.7	8.2 / 0.9	8.9 / 0.8	8.7 /1.2
F	Work Team Assessment / Std. Deviation	8.5 / 0.5	9.1 / 0.7	9.1 / 0.6	9.4 / 0.5
G	Expert Assessment / Std. Deviation	8.0 / 0.7	8.3 / 0.5	7.9 / 0.3	8.2 / 0.4
H	Team Members Visibility / Std. Deviation	8.7 / 0.4	8.5 / 0.5	9.2 / 0.4	9.1 / 0.6
I	Clients-Users Visibility / Std. Deviation	9.1 / 0.3	8.0 / 0.4	9.3 / 0.5	9.2 / 0.3

On the other hand, SSP has been applied to three projects out of the university scenario, in a small software company. They were two SSL and one SSM project. The obtained results reported by the project manager were similar to those shown on Table 1.

The main strengths of SSP are their simplicity and clarity about roles to play (including the client/user), tasks to be done and interactions between activities. These interactions, allows team members to work in an asynchronous and distributed way.

Observing the results we can say that SSP provides a good visibility of the project for both developers and users/clients and produces predictable results. These features make SSP appropriate to support developments of small-size software projects in immature scenarios.

6. Conclusions and Future Work

Usually, small-size software projects carried out in immature development scenarios cannot guarantee either the development time and cost or the quality of the final product. The limitations that well-known heavyweight and lightweight software methodologies have to guide developments in such a scenario were presented in section 3.

In order to deal with this problem, authors have studied several software projects in Chile to identify key issues that are the source for most problems. The results showed a poor understanding or consideration of key issues such as: rights and duties of team members' roles, development context, process activities, coordination protocols, users/clients participation and project visibility. SSP has taken these issues into account. It evolutions has been guided by the lessons learned with each project. The results obtained of its application in 22 projects are encouraging. SSP seems to be a viable alternative to guide small-size software development in immature scenarios.

This proposal is based on the cases we had at hand. We do not know yet its extensibility to other cultural settings. This will be the subject of a forthcoming paper. However, it is possible to hypothesize its applicability to similar cultural and economical environments such as other Latin-American countries.

In the short term, we will continue testing SSP in the reported scenario, and we will start testing such a methodology within software companies. In the long term, we plan to use SSP in several software developments settings in order to identify its limitations.

Acknowledgements

This work has been partially supported by grants Nº 1030959 and 1040952 from Fondecyt (Chile).

References

[1] K. Beck. Extreme Programming Explained. Addison-Wesley, 2000.
[2] B. Bohem. A Spiral Model for Software Development and Enhancement. IEEE Computer, Vol. 21, No. 5, 61-72. 1988.
[3] A. Cockburn. Agile Software Development. Adisson-Wesley, 2002.
[4] O. DeTroyer and K. Leune. WSDM: a User Centered Design Method for Web Sites. Proc. of the Int. 7th World-Wide Web Conference, Brisbane, Australia. 1998.

[5] European Space Agency. ESA Software Engineering Standards. PSS-05-0 Issue 2. ESA Board for Software Standardization and Control (BSSC). February, 1991.

[6] M. Fowler. The New Methodology. April 2003. http://www.martinfowler.com/articles/newMethodology.html.

[7] M. Gaedke, D. Schempf and H. Gellersen. WCML: An Enabling Technology for the Reuse in Object-Oriented Web Engineering. Proc. of 8th Int. World Wide Web Conference (WWW8). Toronto, Ontario, Canada. 1999.

[8] M. Gaedke and G. Graef. Development and Evolution of Web-Applications using the WebComposition Process Model. Proc. of Int. Workshop on Web Engineering at the WWW9, Amsterdam, The Netherlands, May 2000.

[9] F. Guerrero. Success Factors for Adopting and International Process Standard in a Chilean Software Organization: An Experimental Study. Master Thesis. DCC. Universidad Catolica de Chile. Santiago, Chile. May, 2003.

[10] IDC Chile. The Chilean Software Industry. A Study for Japan External Trade Organization (in Spanish). International Data Corporation Chile (IDC Chile). 2003.

[11] R. Jeffries, A. Anderson, and C. Hendrickson. Extreme Programming Installed. Addison-Wesley, 2001.

[12] P. Kruchten. The Rational Unified Process- An Introduction. Third Edition. Addison-Wesley. 2004.

[13] J. Nielsen. Usability Engineering. Academic Press, London, 1993.

[14] S. Ochoa, M.C. Bastarrica. CWADEE: A Chilean Web Application Development Effort Estimation Process. In Proceedings of LA-Web 2003 Conference. IEEE Press. Santiago, Chile. 10-12 November, 2003.

[15] S. Palmer and M. Felsing. A Practical Guide to Feature-Driven Development. Prentice Hall, 2002.

[16] R. Pressman. Software Engineering: A Practitioner's Approach. 5th Edition, McGraw Hill. 2000.

[17] E. Sacre. A Methodology To Develop Web Applications in Small and Medium Size Enterprises (in Spanish). Master Thesis. Computer Science Department, University of Chile. June, 2003.

[18] D. Schwabe, G. Rossi, S, Barbosa. Systematic Hypermedia Design with OOHDM. Proceedings of the International Conference on Hypertext' 96. Washington, USA. 1996.

[19] K. Schwaber. The Scrum Development Process. Proceedings of OOPSLA '95, Workshop on Business Object Design and Implementation, Austin, Texas, USA. ACM Press. October 1995.

[20] D. Schwabe, G. Rossi, S, Barbosa. An Object Oriented Approach to Web-Based Applications Design. In: TAPOS – Theory And Practice of Object Systems, 207-225. 1998.

[21] W. Stein. A Web Software Process for Small or Medium-Sized Projects Focused on the Chilean Scenario. Engineering Thesis (In Spanish). Computer Science Department, Universidad de Chile, April 2003.

[22] S. Zapata, M. Lund. Proposal to Measure Software Customers Satisfaction. Proceedings of 1st Argentine Symposium on Software Engineering (ASSE' 2000), Tandil, Argentina. pp.185-197. September 4-9. 2000.